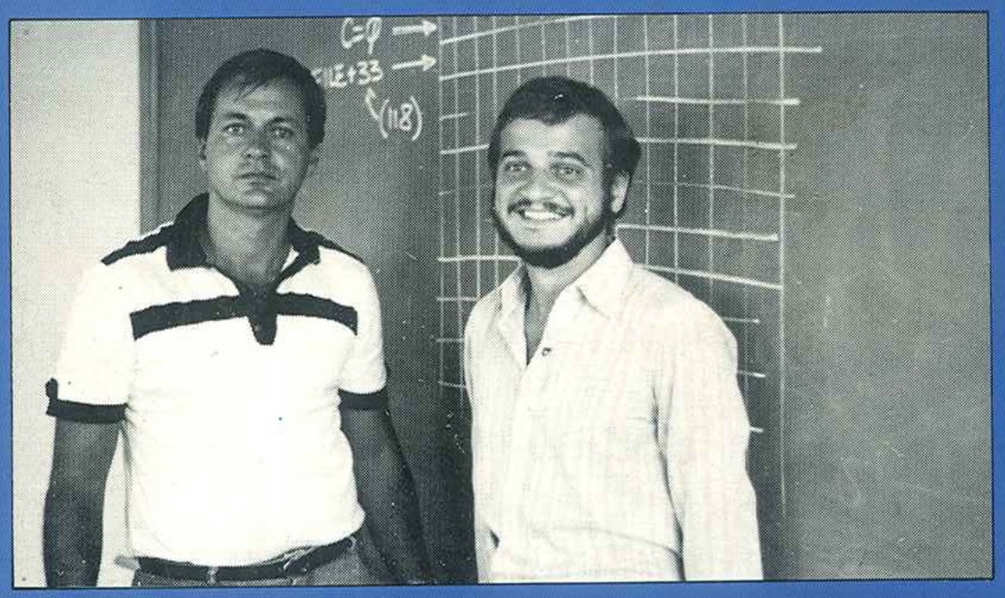
vol. I elementar

Neste volume o leitor toma um primeiro contato com um dos computadores de maior sucesso no Brasil e no mundo.

O texto é extremamente didático e permite que o leitor se alfabetize em BASIC, aprendendo todos os comandos e funções.



PIERLUIGI PIAZZI, físico, químico e professor há 25 anos.

FLAVIO ROSSINI, engenheiro especialista em automação e transmissão de dados por fibras ópticas.

Encontram-se pela primeira vez no ANGLO vestibulares em 1976 como professor e aluno, discutindo ENCONTRO COM RAMA de ARTHUR C. CLARKE (existe força centrifuga?)

Em 1982, FLAVIO, já lidando com linguagem de computação, aprende o BASIC SINCLAIR com PIERLUIGI num curso intensivo (27 minutos) e se torna professor do NÚCLEO DE ORIENTAÇÃO DE ESTUDOS, cujas apostilas acabaram gerando uma série de livros de sucesso.

Da colaboração dos dois surgiu o BASIC TK, do qual este livro é um dos 3 volumes já de há muito pedido pelos leitores, e muitas vezes adiado por um longo trabalho do FLAVIO nos Estados Unidos.

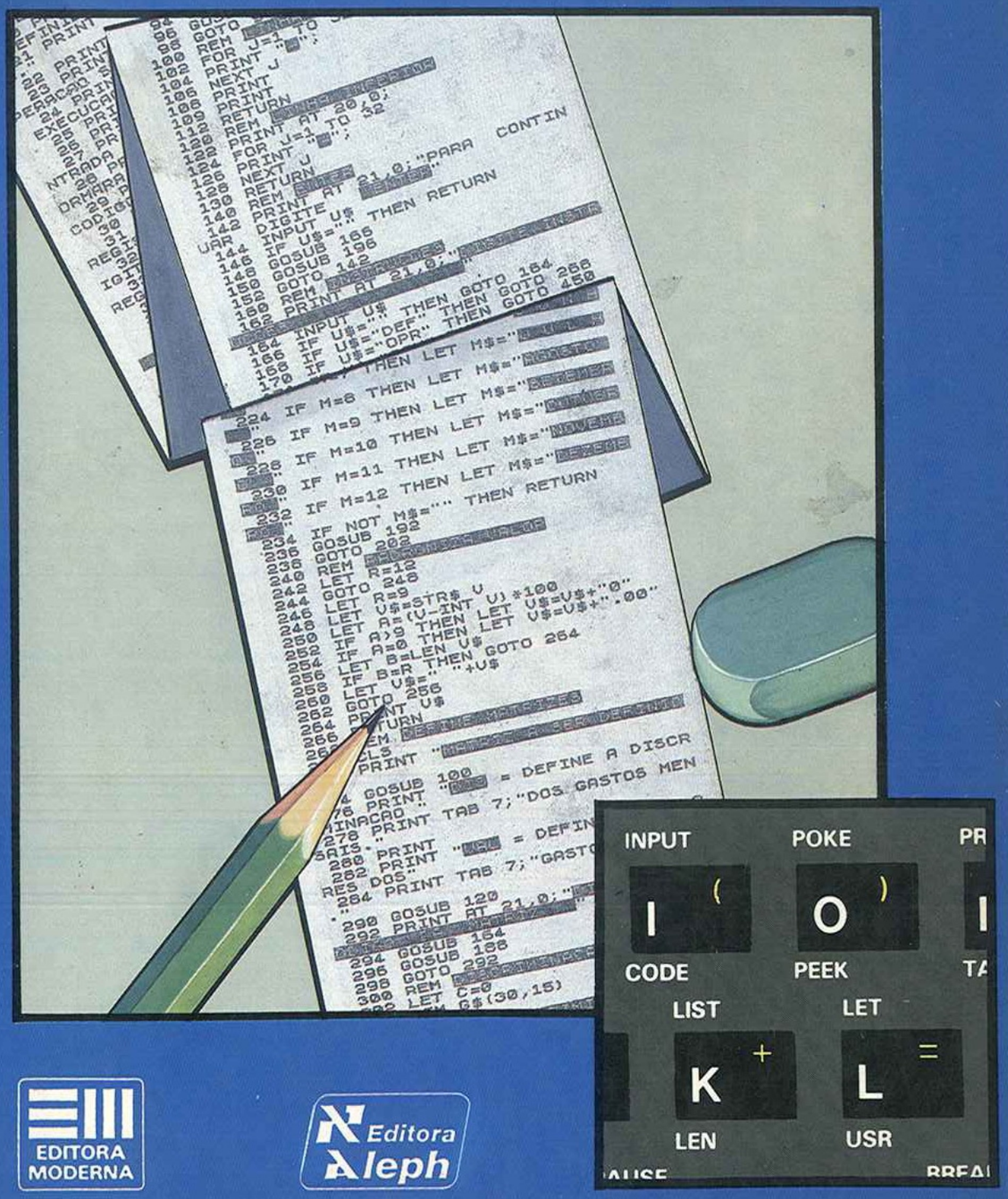
Desta colaboração mais um resultado pronto e uma certeza: força centrífuga **não** existe!

BASIC TK vol. I elementar piazzi-rossini

BASIC TK

TK 82 83 85 RINGO CP 200
SINCLAIR ZX 81 TS 1000

vol. I elementar piazzi-rossini



EDITORA MODERNA

Editora Aleph

BASIC TK

vol. I

elementar

pierluigi piazzi
flavio rossini

BASIC TK

vol. I

elementar

Editora Aleph

EDITORA MODERNA

Diretor Editorial:
Pierluigi Piazzi
Coordenação Didática:
Betty Fromer Piazzi
Ilustração da Capa:
Fátima M. Rossini Gouveia
Capa:
Fernando Moretti
Produção:
Rosa Kogan Fromer
Organização Editorial:
Rosana de Angelo
Arte 3ª Edição:
Ana Lúcia Antico

Editora Aleph

ALEPH PUBLICAÇÕES E
ASSESSORIA PEDAGÓGICA LTDA.
Av. Brig. Faria Lima, 1451 - conj. 31
01451 - São Paulo - SP
tel.: (011) 813-4555

Distribuição exclusiva em livrarias

EDITORA MODERNA

EDITORA MODERNA
R. Afonso Brás, 431
04511 - São Paulo - SP
tel.: (011) 531-5099

1985
3.ª edição
Brasil

**CIP-Brasil. Catalogação-na-Publicação
Câmara Brasileira do Livro, SP**

Piazzi, Pierluigi, 1943—
P647c Curso de BASIC-TK / Pierluigi Piazzi, Flávio Rossini —
v.1 São Paulo : Aleph : Ed. Moderna, 1985

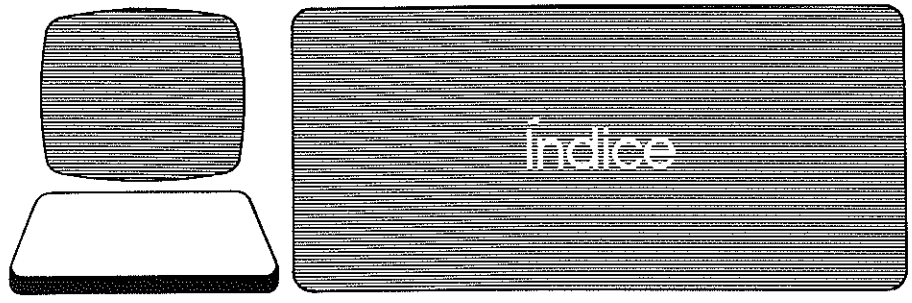
1. BASIC (Linguagem de programação para computadores)
2. Microcomputadores — Programação 3. TK (Computador) —
Programação I. Título

17 CDD-651.8
18 -001.6424
85-0571 18 -001.642

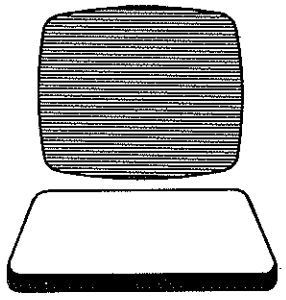
Índices para catálogo sistemático:

1 BASIC. Linguagem de programação : Computadores : Processamento de dados 651.8 (17) 001.6424 (18.)
2 Microcomputadores : Linguagem de programação : Processamento de dados 651.8 (17) 001.642 (18.)
3 Microcomputadores : Programação : Processamento de dados 651.8 (17) 001.642 (18.)
4 Programação : Microcomputadores : Processamento de dados 651.8 (17) 001.642 (18.)
5 TK : Computadores : Programação : Processamento de dados 651.8 (17) 001.642 (18.)

Agradecimentos à
Betty
e
Waléria
pelo amor e compreensão...



Índice



Prefácio

Temos visto surgir no mercado um bom número de cursos de BASIC bem como manuais sendo escritos com a finalidade de ensinar seu possuidor não só os comandos do computador como também fundamentos da linguagem BASIC.

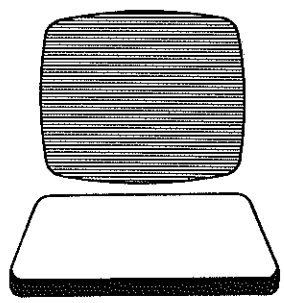
A rapidez com que estes volumes desaparecem das estantes das livrarias, apesar da grande quantidade de títulos, mostra como eles são necessários.

Mais um livro na área seria bem-vindo. Mas o BASIC TK não é apenas mais um livro. Não faria sentido num mar de livros surgir mais um que não tivesse nada de especial a oferecer. O que o diferencia dos outros é que ele é dirigido aos possuidores de computadores TK e seus compatíveis (CP-200, NE Z8000 e ZX81), que são, no momento, os computadores mais vendidos no Brasil.

Além disso, o livro dos professores Flavio Rossini e Pierluigi Piazzi é bastante didático, complementando o manual com explicações mais detalhadas, de modo que os usuários destes computadores possam tirar o máximo proveito de suas máquinas.

Por outro lado, apesar de o livro ser dedicado aos TKs e similares, o texto pode ser encarado como um livro de apoio para qualquer computador, desde que seja programável em BASIC e o leitor tenha em mente as peculiaridades tanto de seu computador como do TK.

Alvaro A. L. Domingues



Introdução

Este livro apresenta a linguagem BASIC na forma utilizada pelos computadores compatíveis com os TK-82, TK-83, TK-85, CP200, etc., por nós chamada de BASIC-TK. Ela é aqui colocada "passo a passo", de um modo bastante acessível de maneira que possa ser facilmente entendida, mesmo por quem não possua nenhuma noção sobre programação de computadores. O objetivo deste volume é cobrir todas as características do BASIC-TK ilustrando seu funcionamento através de exemplos e exercícios e dando uma pequena noção de estrutura de programação, além de uma leve "pincelada" em linguagem de máquina. O leitor interessado poderá encontrar, no segundo volume desta série, aplicações mais "profundas" do BASIC para jogos, finanças e administrações, com bastante ênfase à estrutura de programação. Para os que iniciam agora o aprendizado desta maravilhosa ciência, bem-vindos ao mundo da informática.

1 CAPÍTULO

Apresentação: digitação e noções básicas

O que é um computador?

Um computador é uma série de circuitos eletrônicos capazes de interpretar e executar ordens recebidas dos seres humanos para realizar operações aritméticas (e/ou lógicas), funções matemáticas e manusear informações, podendo inclusive controlar outros aparelhos.

O computador se caracteriza por:

a) ser RETARDADO

b) ter MUITA MEMÓRIA, sendo incapaz de ESQUECER alguma coisa (enquanto estiver ligado!)

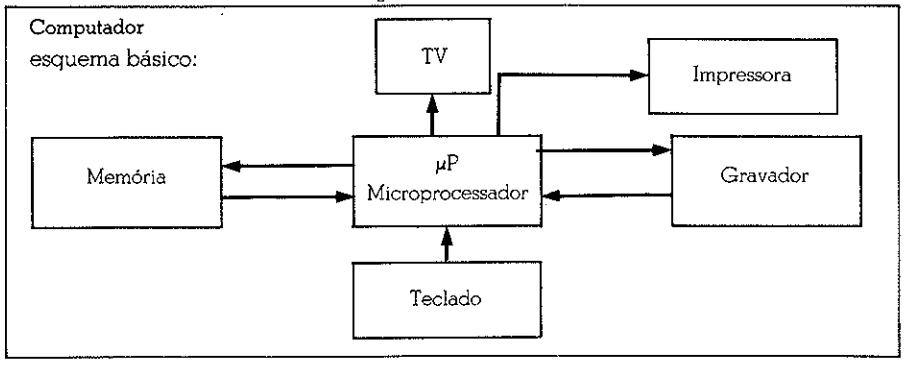
c) ser MUITO RÁPIDO

d) ser PRECISO

e) não ERRAR nunca (a não ser que você mande.)

f) ser EXTREMAMENTE OBEDIENTE, executando ordens sem qualquer reclamação!

Os computadores, analogamente aos seres humanos, têm um cérebro (responsável pelo controle e cálculo) e dispositivos para se comunicar com o mundo externo, ou seja, receber informações e enviar informações. No caso do TK, o esquema básico seria o que mostramos:

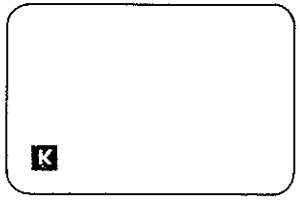


O microprocessador é o cérebro, usando a memória para armazenar dados e informações, o teclado para receber e a TV ou impressora para enviar informações ao ser humano. Como o computador só é capaz de reter estes dados enquanto estiver "acordado" (ligado), ao "adormecer" (ser desligado) ele esquece TUDO o que havia sido colocado na memória! Portanto utiliza-se um gravador K-7 para gravar informações que eventualmente serão utilizadas de novo no futuro.

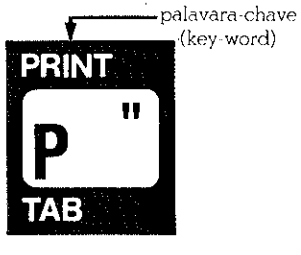
Note que de todas as características do computador, a mais indesejável é a primeira, ou seja, o fato dele ser RETARDADO! E é justamente isto que devemos ter sempre em mente; o computador é rápido, obediente e preciso mas NÃO tem inteligência; apenas se limita a executar ordens EXATAMENTE como foram formuladas.

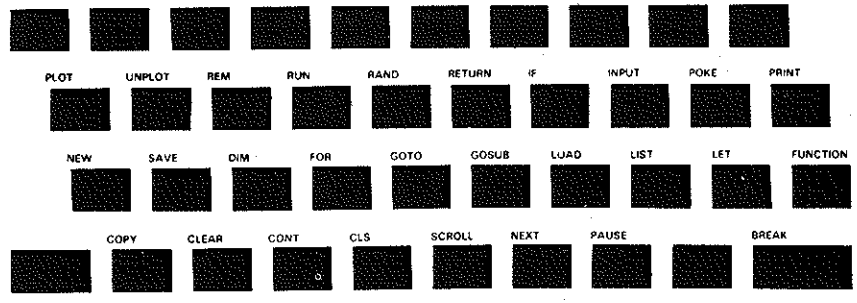
O Cursor K

Ligue então o seu TK, colocando a chave (ON-OFF) da fonte de alimentação na posição ON. No canto inferior da tela deve aparecer um quadradinho com o K em seu interior.



Ele é chamado de cursor e será muito útil para nos ajudar a trabalhar com o computador. Este K é abreviação de KEYWORD, ou seja, palavra-chave. As palavras-chaves são as que aparecem ACIMA de cada tecla; assim, se o cursor estiver em K ao apertar qualquer tecla que tenha uma palavra-chave acima da mesma, esta palavra aparecerá no canto inferior esquerdo da tela.

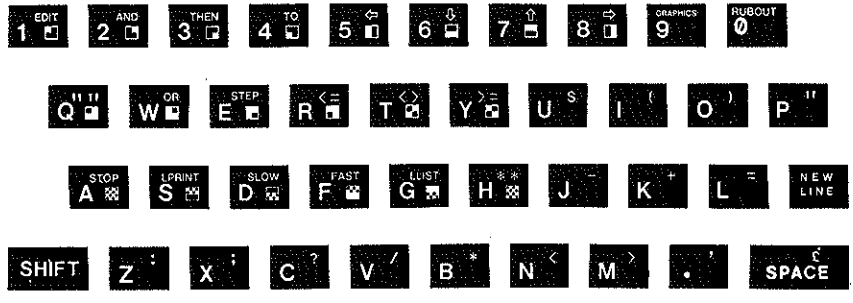




as "key-words" do TK

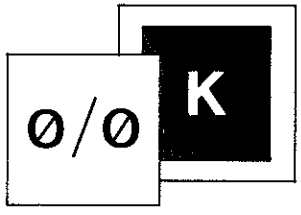
A instrução PRINT — Os cursores L S e F

Vamos começar a dar ordens ao computador; aperte a tecla da letra P; você irá obter PRINT na tela, seguido do cursor que agora mudou para L (que e a abreviação de LETTER, ou seja, letra), indicando que o computador não mais espera uma palavra-chave, mas sim uma letra, número, ou símbolos especiais (+, -, *, <, etc...) como se fosse uma máquina de escrever convencional; em outras palavras, com o cursor em L você pode acessar tudo o que está desenhado "dentro" das teclas menos os caracteres gráficos (▚ ▄ ▙, etc...); os símbolos em vermelho no TK82 e TK83 (ou em amarelo no TK85) podem ser acessados pressionando simultaneamente a tecla SHIFT e a tecla desejada.



as "letters" do TK

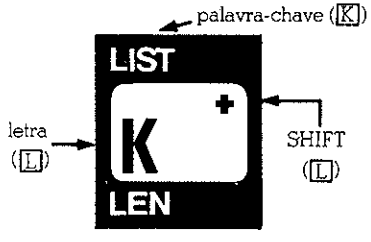
Bom, após ter colocado PRINT na tela, experimente digitar o número 67. O que você está ordenando? Ora, PRINT siginifica ESCREVA, portanto, você está dizendo ao computador: "Escreva o número 67". Para ele executar a sua ordem, basta informá-lo de que ela está completa e isto pode ser feito pressionando a tecla NEW LINE. Verifique o que aparece no canto superior esquerdo. No canto inferior aparece o Código Ø/Ø; não se importe com isto por enquanto; apenas saiba que o cursor K está "escondido" detrás destes números.



De fato pressione P e você obterá novamente PRINT; faça então com que ele escreva o resultado de uma conta de somar, por exemplo:

```
PRINT 7+9
```

Note que o caractere "+" está na tecla da letra K. Para fazer com que o "+" apareça na tela, basta então, apertar a tecla SHIFT; e, simultaneamente, a tecla que contém a letra K.

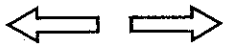


Se você esquecer disto, você irá obter:

```
PRINT 7K9
```

que não faz nenhum sentido! Teremos que, então, apagar a letra K. O computador é capaz de apagar somente o caractere que estiver imediatamente à esquerda do CURSOR; portanto, devemos aprender a deslocá-lo.

Observe as teclas 5 e 8; nelas você verá, em vermelho(ou amarelo), o seguinte:



portanto, ao apertar SHIFT e, simultaneamente, as teclas 5 ou 8, o cursor caminhará para a esquerda ou para a direita. Suponha que você tivesse cometido o erro acima mencionado estando com o cursor na posição indicada. Pressione SHIFT e o 5 ao mesmo tempo e você obterá:

```
PRINT 7K9
```

Pronto, o caractere que desejamos apagar está imediatamente à esquerda do cursor; para apagá-lo basta pressionar SHIFT e a tecla Ø (RUBOUT) simultaneamente; você obterá:

```
PRINT 79
```

Agora coloque o sinal de "+" e a seguir pressione NEW LINE; o computador executa sua ordem e escreverá 16 no topo da tela. Se você mandar o computador executar uma ordem que não tem sentido, duas coisas podem ocorrer: ou ele escreve uma mensagem de erro no canto inferior esquerdo da tela (explicaremos isto logo mais) ou ele aponta na própria ordem com o cursor S(de sintaxe)onde está o erro; isto depende do tipo de erro que você cometer!

Experimente colocar:

```
PRINT +-8
```

e a seguir NEW LINE. (O caractere "—" está na tecla J)

Você obterá:

```
PRINT +-8
```

Desloque então o cursor L para a direira do sinal de "+" e apague-o; substitua-o pelo número 3:

```
PRINT 3-8
```

e pressione NEW LINE. Agora o computador é capaz de entender a sua ordem e executará a operação, fazendo aparecer—5 na tela.

A outra maneira do TK apontar erros é através de mensagens que aparecem no canto inferior esquerdo da tela. Agora há pouco vimos o código Ø/Ø logo após executar ordem de escrever o número 67 na tela; o primeiro número indica o tipo de erro que cometemos: no caso, o código Ø significa tudo O.K., não

houve erros. Se este número for diferente de 0, significa que cometemos algum erro. Ao longo do livro, procuraremos apresentar as várias mensagens de erro que o TK utiliza. No entanto, elas se encontram detalhadas no manual do TK. Quanto ao segundo número que aparece, não se preocupe com ele, por enquanto.

Vamos agora ordenar que o computador calcule o valor de uma expressão aritmética:

```
PRINT 10-2*5**2+3/5*2
```

Neste ponto, várias observações são necessárias: não confunda a letra O com o número zero, que é assim representado: 0 . O computador interpreta o símbolo "*" como um sinal de MULTIPLICAÇÃO, o símbolo "/" como um sinal de DIVISÃO o símbolo "**" (tecla H) como um sinal de EXPONENCIAÇÃO. A ordem de prioridade é:

```
PRIMEIRA PRIORIDADE -> **
SEGUNDA  PRIORIDADE -> *  E  /
TERCEIRA PRIORIDADE -> +  E  -
```

Assim, esta expressão será executada desta maneira:

```
10-2*5**2+3/5*2=
10-2*25+3/5*2=
10-50+3/5*2=
10-50+0.6*2=
10-50+1.2=
-40+1.2=
-38.8
```

(em caso de empate, a 1ª operação a ser executada será a que estiver mais a esquerda) e o -38.8 aparecerá na tela. Note que o computador usa "." (ponto) ao invés da "," (vírgula), segundo a notação inglesa.

Verifique que, o sinal de exponenciação ** deve ser o da tecla onde está a letra H; não adianta pressionar duas vezes o sinal * da tecla B! Apesar do efeito na tela ser o mesmo, o computador não entenderá sua ordem e dará um erro de sintaxe!

Numa expressão aritmética você pode também utilizar os parênteses (letras I e O), por exemplo:

```
PRINT 4*(7-2)
```


Neste caso, a subtração será executada primeiro e você obterá o número 20.

Surge agora a seguinte pergunta: e se eu desejar escrever uma palavra ou frase? É simples! Basta colocar a palavra ou frase entre aspas (", tecla P), por exemplo:

```
PRINT "BEATLES"
```

e, a seguir, NEW LINE!

Note que as aspas significam o seguinte: "computador, copie exatamente o que estiver escrito entre aspas"! De fato, experimente colocar:

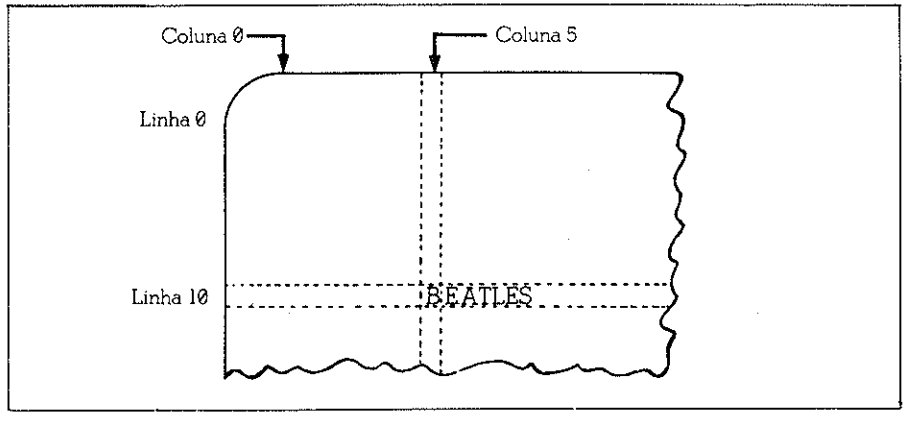
```
PRINT "3+4"
```

e NEW LINE. Sua ordem será executada; ele copiará 3 + 4 no topo da tela mas SEM executar a operação. Portanto, você poderá colocar todos os caracteres que quiser (letras, números, símbolos) entre aspas. Se você desejar que as próprias aspas sejam copiadas no topo da tela, você deverá usar as aspas duplas (tecla Q), da seguinte maneira:

```
PRINT "  ""BEATLES"" "
```

pressione NEW LINE e você irá obter "BEATLES", ao invés de BEATLES, no canto superior da TV.

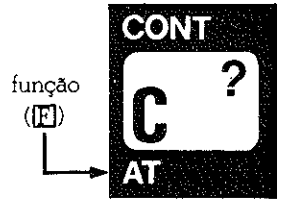
A esta altura, você deve estar cansado(a) de escrever sempre no topo da tela. Por isso, vamos variar um pouco. A tela tem 22 linhas e 32 colunas disponíveis numeradas de 0 a 21 e 0 a 31, respectivamente, sendo que a posição do canto superior esquerdo da tela é a posição 0,0. Note que estas 22 linhas não incluem as linhas onde você escreve a sua ordem (você tem 2 linhas para isto). Vamos então escrever BEATLES a partir da linha 10 e coluna 5:



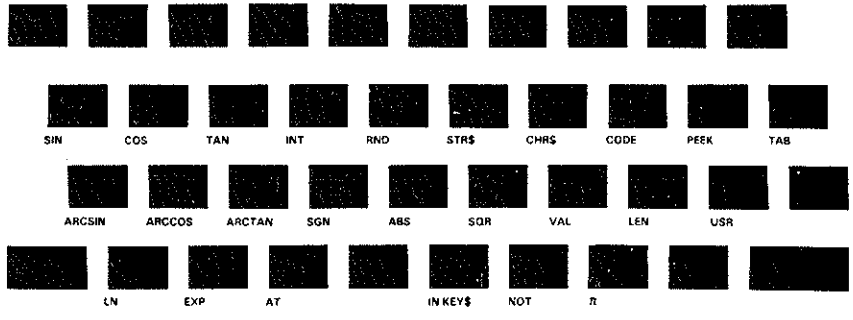
Basta fazer o seguinte:

```
PRINT AT 10,5;"BEATLES"
```

Preste atenção: a palavra AT não é uma palavra-chave, nem letra mas sim uma função e,para que possamos escrevê-la,devemos mudar o cursor de L para F.Para fazer isto, pressione SHIFT e NEW LINE (FUNCTION) simultaneamente. Agora, com o cursor em F, aperte a tecla C. Todas as palavras que estiverem abaixo de cada tecla são tratadas como FUNÇÕES e, para obtê-las na tela, é necessário o cursor estar em F.



Não ADIANTA tentar escrever a função; por exemplo, no caso, apertando a letra A e a letra T e a seguir um espaço . Experimente fazer isso e veja o que ocorre!



as "functions" do TK

Faça o seguinte:

```
PRINT AT 12,28;"BEATLES"
```

Tente explicar o que acontece.

A instrução LET: variáveis numéricas na memória

Experimente, agora, tentar escrever uma palavra sem usar as aspas na ordem PRINT.

```
PRINT LOVE
```

Ao apertar NEW LINE, nada aparecerá no canto esquerdo superior da tela, mas também não aparecerá o cursor S indicando erro; temos apenas a mensagem 2/0 onde o número 2 indica que "alguma coisa está errada".

Veja porque: quando uma letra ou grupo de letras aparece sem aspas, o computador interpreta esta letra ou grupo como uma variável numérica , cujo valor deveria estar em sua memória; ele irá procurá-la sem no entanto encontrá-la e por isso não poderá escrever seu valor. Experimente, então, fazer:

```
LET LOVE=10 (LET->TECLA L)
```

e, a, seguir, NEW LINE. Nada aparecerá novamente no topo da tela. De fato, você não ordenou ao computador que ele escrevesse mas sim que, em sua MEMÓRIA, atribuísse à variável LOVE o valor 10!

Experimente fazer agora:

```
PRINT LOVE
```

O que você obtém? Ora, ele irá procurar LOVE em sua memória e irá achar o número 10 que será então escrito.

Note que a mensagem agora é 0/0 indicando que tudo está Ok. Portanto o código 2 indica que você tentou acessar uma variável na memória cujo valor era desconhecido pelo computador. A instrução LET faz então com que o "espaço de memória" seja reservado para uma dada variável e a ela atribuído um valor.

Faça agora:

```
LET LOVE=LOVE+1
```

Que valor será agora atribuído à variável LOVE na memória do TK?

Note que o sinal de igual usado no computador não tem o significado usual: de fato, seria absurdo matematicamente dizer que um número é igual a ele mesmo mais um. Portanto o sinal de igual significa: atribua à variável da esquerda o número ou resultado da operação que estiver à direita.

Digite agora:

```
PRINT "LOVE";LOVE
```

Note bem o efeito das aspas. O ponto e vírgula é necessário para o computador entender que deve escrever coisas distintas. Se você não gostou do fato dos escritos saírem "grudados", basta colocar um espaço em branco após a palavra LOVE entre aspas; para obtê-lo, pressione a tecla SPACE;

```
PRINT "LOVE ";LOVE
```

Lembre-se, na memória do computador agora está a variável LOVE associada ao número 11. Obviamente se você desligá-lo ele irá "esquecer" isto e ao ser ligado novamente nada aparecerá na tela se você ordenar:

```
PRINT LOVE
```

Entretanto, você pode "apagar" as variáveis da memória do computador sem desligá-lo, usando a ordem (palavra-chave CLEAR(letra X)):

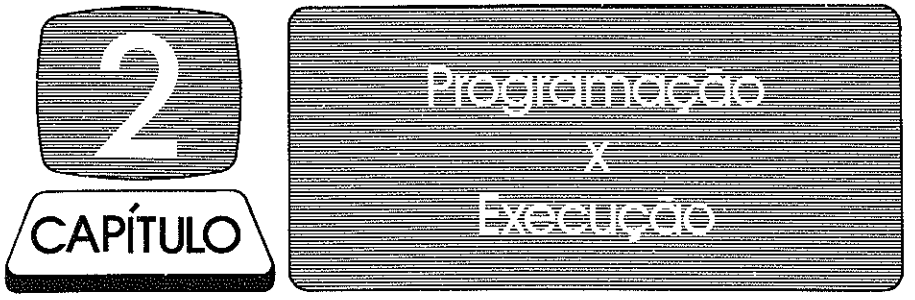
Experimente então fazer CLEAR, NEW LINE e, a seguir, faça:

```
PRINT LOVE
```

Como esperado, nada será escrito, pois o computador "limpou" a sua memória!

Note novamente o código 2/Ø.

Neste primeiro capítulo a quantidade de informações apresentada é muito grande, e o mesmo acontecerá no segundo capítulo. Isto assemelha-se a quando nós começamos a aprender uma nova língua, onde tudo parece complicado até o momento em que finalmente começamos a fazer uma frase. Portanto, não desanime! Estude com vontade e logo você estará "programando" o computador...



As palavras-chaves RUN E LIST — Os cursores ▶ e G

Até agora, só mandamos o computador executar uma ordem por vez; mas como fazer para que ele execute várias ordens em seqüência? Para isso é necessário *numerar* as ordens (também chamadas instruções). Por exemplo, digite:

```
10 LET A=Ø
```

e, a seguir, NEW LINE. Pelo fato de ter um número na sua frente, a ordem não é executada; apenas copiada na parte de cima da tela da seguinte maneira:

```
10 LET A=Ø
```

a seguir, digite:

```
20 PRINT A
```

e NEW LINE, na tela você irá obter:

```
10 LET A=Ø
20 PRINT A
```

Notamos que apareceu um novo cursor, o cursor ▶ que indica para qual das linhas o computador está "olhando". Para deslocá-lo utilize SHIFT juntamente com as teclas 6 (↓) e 7(↑). Suponha que você tenha "errado" a instrução (ou linha) 10 e quisesse fazer com que a variável A fosse igual a 15 e não Ø. Desloque então o cursor para a linha 10 (usando SHIFT e a tecla 7); para poder corrigi-la, devemos deslocar o cursor usando (ou→(ou←), mas isto só é possível se a linha estiver na parte de baixo da tela. Para trazê-la a esta posição, basta apertar SHIFT e a tecla 1 (EDIT) e você obterá a linha copiada na parte inferior da tela:

```
10 LET A=Ø
```

Desloque o cursor até depois do dígito Ø, apague-o usando (RUBOUT) e substitua-o por 15, você obterá:

```
10 LET A=15
```

e, a seguir, NEW LINE, a linha 10 no topo da tela será substituída por esta NOVA linha 10. Vamos fazer voltá-la à original, usando agora outro método; sem trazer a linha para baixo, escreva uma nova linha 10 na parte inferior da tela:

```
10 LET A=0
```

e aperte NEW LINE.
Novamente a linha 10 superior será alterada.
Para exercitar, tente agora substituir a linha 10 por:

```
10 LET A=5
```

utilizando qualquer um dos dois métodos.

Feito isso, vamos acrescentar outra instrução:

```
30 PRINT 3*A
```

Após apertar NEW LINE, teremos então o seguinte:

```
10 LET A=5
20 PRINT A
30 PRINT 3*A
```

Observe que, intencionalmente, não estamos utilizando números consecutivos para numerar as instruções. Isto é útil para podermos colocar novas instruções entre duas já existentes, o que não seria possível se as duas tivessem números consecutivos. Por exemplo, digite o seguinte:

```
25 PRINT "A=";A
```

e a seguir NEW LINE. Você observará que esta instrução "toma" seu devido lugar entre as linhas 20 e 30. O que temos agora? Vejamos, 4 instruções "numeradas"; se executássemos uma de cada vez, iríamos obter, respectivamente, na tela: nada na primeira, pois ela apenas atribui o valor 5 na memória à variável A; na segunda, apareceria o número 5 na tela; na terceira, a letra A seguida do caractere = e do valor da variável A (ou seja 5) e na quarta, o número 15 (resultado da operação 3*5).

Agora, poderemos executá-las todas sequencialmente. Para isto basta pressionar a tecla R; como o cursor está em [K], aparecerá a palavra-chave:

```
RUN
```

e a seguir NEW LINE. Esta instrução tem o seguinte efeito: faz inicialmente um CLEAR (para limpar todas as variáveis que eventualmente, tinham sido definidas), e a seguir, ordena ao computador que ele execute sequencialmente as ordens numeradas. Portanto na tela você irá obter:

```
5
A=5
15
```

e na memória teremos o número 5 associado à variável A. Note o efeito do ponto e vírgula (;) na instrução 25. Ele indica que duas coisas serão escritas "lado a lado", com um só PRINT. De fato, durante a execução de um PRINT após o

outro faz com que o segundo escreva sua linha imediatamente abaixo do primeiro desde que não haja um ponto e vírgula .

Preste atenção: quando você numera as ordens o computador NÃO as esquece. Em outras palavras, as ordens numeradas são armazenadas em sua memória. Neste instante, você não as vê na tela pois ele está apenas mostrando os resultados das ordens. Para fazer com que ele mostre as ordens na tela, utilize a seguinte instrução:

```
LIST          (LETRA K)
```

e NEW LINE. O que acabamos de fazer é muito importante e precisa estar bem claro em sua cabeça: ao ordenar ao computador que faça uma determinada tarefa, *sem* colocar um número na frente da ordem, o computador executa *imediatamente* esquecendo a ordem. Se você numerar as ordens, elas serão executadas ao receber o comando RUN e o computador *não* as esquece. Elas estão em sua memória e podem ser vistas na tela fazendo um LIST. A fase em que você escreve as ordens no computador é chamada de *programação*; ao apertar a tecla RUN, ele executará as ordens do seu programa, fase essa que chamamos de *execução*. É importante não confundir as duas coisas.

Vamos agora "brincar" um pouco com o nosso programa, acrescentando a instrução:

```
40 LIST
```

e execute este "novo programa", apertando RUN e NEW LINE. Veja, o computador irá executar as suas ordens, sendo que a última, diz para que ele *mostre* o programa na tela, assim você irá obter:

```
5
A=5
15
   10 LET A=5
   20 PRINT A
   25 PRINT "A=";A
   30 PRINT 3*A
   40 LIST
```

Ou seja, no programa você ordenou que ele "se mostrasse" no final. Substitua agora a linha 40 por:

```
40 RUN
```

E execute o programa, fazendo RUN e NEW LINE.

O que você obtém? Várias vezes a mesma saída 5, A e 15. Você saberia explicar por quê? Observe a sua última instrução. Nela você, ao terminar o programa, ordena que ele comece de novo. Dessa forma, ele irá se repetir até não haver mais espaço na tela. Verifique que no canto inferior esquerdo da tela temos o código 5 indicando que não há mais espaço na tela.

Em todos estes exemplos, a ordem de RUN indica ao computador que ele

comece a executar o programa, a partir da primeira linha. Você pode fazer com que ele execute, a partir de qualquer outra linha. Faça LIST e NEW LINE e, a seguir, substitua a linha 40 por:

```
40 RUN 30
```

a seguir digite RUN e NEW LINE.

Veja o que acontece: o computador "escreve" na tela: 5, A e 15 apenas uma vez, pois RUN 30 significa "dê um CLEAR", e execute o programa a partir da linha 30. Ora, ao tentar executar a instrução 30 ele irá procurar a variável A na memória, mas não irá encontrá-la, pois ela foi apagada pelo CLEAR; assim, ele pára a execução do programa. Experimente, então, substituir a linha 30 por:

```
30 LET A=8
```

e acrescente a linha 35 da seguinte maneira:

```
35 PRINT 3*A
```

Execute então o programa assim modificado. Tente explicar o que ocorre.

```
10 LET A=5
20 PRINT A
25 PRINT "A=";A
30 LET A=8
35 PRINT 3*A
40 RUN 30
```

```
 5
A=5
 24
 24
 24
 24
 24
 24
 24
 24
 24
 24
 24
 24
 24
 24
 24
 24
 24
 24
 24
 24
 24
```

Da mesma forma, a instrução LIST sozinha mostra o programa na tela a partir da primeira linha. Experimente ordenar:

```
LIST 25
```

e NEW LINE. Observe que o computador mostra o seu programa na tela a partir da linha 25.

```
25 PRINT "A=";A
30 LET A=8
35 PRINT 3*A
40 RUN 30
```

Acrescente ao seu programa agora, 2 linhas:

```
22 PRINT
27 PRINT
```

de tal maneira que ele fique:

```
10 LET A=5
20 PRINT A
22 PRINT
25 PRINT "A=";A
27 PRINT
30 LET A=8
35 PRINT 3*A
40 RUN 30
```

e mande agora executar o seu programa (RUN e NEW LINE). Você irá obter uma linha em branco, entre o número 5 a letra A, e, entre a letra A e o primeiro número 24, pois, de fato, você ordenou que ele NÃO escrevesse nada, ou melhor, escrevesse uma linha em branco.

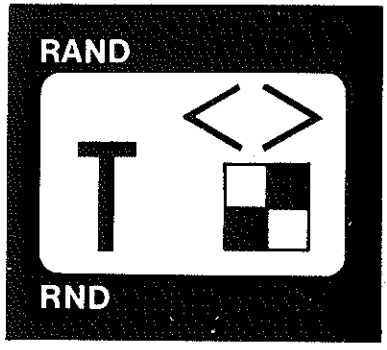
Dessa forma, você conseguiu fazer com que ele "pulasse" linhas.

Cabe aqui um comentário com relação à instrução CLEAR: ela apenas limpa as *variáveis* do computador. De fato, experimente ordenar CLEAR (e NEW LINE) e, a seguir faça LIST. Você pode observar que seu programa ainda está lá.

Lembre-se que para "limpar" (apagar) uma linha de programa, basta colocar o número da linha, seguido de NEW LINE; experimente escrever 40 e NEW LINE, a linha 40 irá desaparecer. Se você quiser apagar todo o programa e todas as variáveis, basta executar o comando NEW (tecla A) e NEW LINE.

Experimente "listar" o seu programa; nada aparecerá na tela, ele "esqueceu" tudo... Portanto, cuidado com este comando; ele é EXTREMAMENTE PERIGOSO, pois causa amnésia em seu computador.

Para finalizar, vamos aprender mais um cursor: trata-se do cursor G (GRAPHICS) cujo acesso é feito pressionando SHIFT e a tecla 9. Observe, por exemplo, a tecla T, que mostramos na figura:



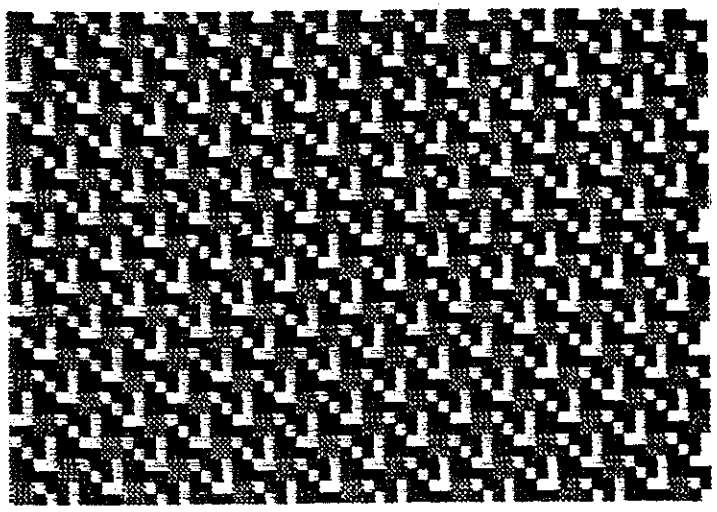
Se o cursor estiver em K, ao pressioná-la você obtêm RAND na tela; com o cursor L, você terá T, ou < > se pressionar simultaneamente o SHIFT; com o cursor em F, você obtém RND; agora, com o cursor em G ao pressioná-la, você terá a letra T em vídeo reverso, ou, o desenho ▚, se pressionar simultaneamente o SHIFT (a palavra-chave RAND, a função RND e o símbolo < > serão discutidos mais adiante.)
Façamos o seguinte programa:

```
10 PRINT "▚▀▀▌"
20 RUN
```

Note que, após abrir aspas, devemos colocar o cursor em G e, simultaneamente com SHIFT, apertar as teclas T, H, 7, F e 8. Para fechar as aspas, devemos passar novamente o cursor para L. Para isto, basta pressionar SHIFT e a tecla 9 (GRAPHICS). Execute então o programa. A seguir, modifique a linha 10 acrescentando um ponto e vírgula (;) no final:

```
10 PRINT "▚▀▀▌";
20 RUN
```

Execute o programa e perceba o efeito "*estranho*" do (;) ou seja: faça com que o próximo PRINT escreva "ao lado" e não na linha de baixo. Mais adiante veremos alguns detalhes a respeito do ponto e vírgula (;).



Memória: 2k, 16k, 48k, 64k...

Vamos aqui fazer uma pequena observação quanto à memória do computador. A unidade de medida para memória é o *byte*. O TK82C e o TK83 vêm originalmente com 2k bytes (2048 bytes*) de memória disponível e o TK85 tem 16k bytes na sua versão mínima. Existem expansões de memória capazes de aumentar a capacidade do TK82 e TK83 até 16k (usando expansão de 16k) ou até 56k (usando expansão de 64k, sendo que apenas 48k desses 56k são acessíveis diretamente pelo BASIC); o TK85 pode ser adquirido também com 48k bytes de memória. O que significa tudo isto? Quanto mais memória você tem, maiores e mais complexos poderão ser seus programas... Neste livro, todos os programas apresentados para ensinar o BASIC TK cabem em 2k apenas; entretanto, para a maioria dos programas aplicativos aqui apresentados, é necessário ter pelo menos 16k de memória. Note que estes programas são apenas um complemento ao seu aprendizado, mas seria conveniente que você os fizesse; assim, se você tem um TK82 ou TK83 mas não tem expansão, veja se consegue arrumar uma emprestada pelo menos para acompanhar o seu estudo desse livro.

* 1k byte é igual a 1024 bytes, ou seja 2^{10} bytes.

EXERCÍCIOS:

1) Faça um programa que associe à variável HOPE o número 6 (na memória) usando a instrução LET e, a seguir, escreva na tela o seguinte:

```
HOPE = 6

HOPE + 2 = 8

3*HOPE = 18
5+7 = 12
5+7 = 12
5+7 = 12
5+7 = 12
5+7 = 12
5+7 = 12
5+7 = 12
5+7 = 12
5+7 = 12
5+7 = 12
5+7 = 12
5+7 = 12
5+7 = 12
5+7 = 12
5+7 = 12
5+7 = 12
5+7 = 12
```

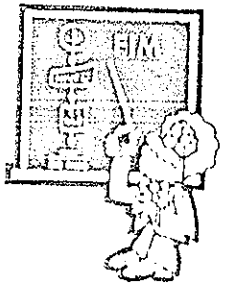
até o fim da tela.

2) "Apague" o programa anterior (NEW) e, a seguir, faça a seguinte instrução (não numerada); LET B = 10 (NEW LINE). Escreva então este programa:

```
100 LET A=5
200 PRINT A
300 PRINT B
```

Note o que aparece na tela ao tentar executá-lo. Explique o que ocorre e a seguir acrescente a linha 150 CLEAR; tente, então, executar o programa e explicar o que acontece. Substitua então a linha 150 por 150 LET B=7 e execute o programa. Finalmente, acrescente a linha 400 NEW.

Execute o programa e tente listá-lo em seguida. Explique o que ocorre.



3 CAPÍTULO

Números Aleatórios e Loops

Números Aleatórios

Vamos aprender agora uma função muito interessante para computador: "chutar" números. Para entendê-la, imagine que dentro do computador existam infinitas cartas, cada carta possuindo um número compreendido entre 0 e 1. Se voce ordenar:

```
PRINT RND
```

O computador escreverá o valor da primeira carta. (Note que RND é uma função e, portanto, você deve mudar o cursor para F e apertar a tecla T). Anote este valor num pedaço de papel. Faça novamente:

```
PRINT RND
```

Anote o novo valor (que corresponde a segunda carta) e repita a operação; você terá então o valor das primeiras três cartas. Desligue o computador, ligue-o novamente e repita a operação; você obterá novamente as primeiras 3 cartas. No entanto, existe um comando que faz com que o computador "chute" qual a primeira carta que ele deve escolher deste monte INFINITO, estando as próximas em seguida; dessa forma, você obterá sempre seqüências diferentes. Este comando é a palavra-chave RAND, que aparece na tecla T, faça:

```
RAND
```

e a seguir execute a operação acima. Desligue e ligue o computador faça RAND e repita. Desta vez os números são diferentes (se, por coincidência do destino,

isto não ocorrer, repita a operação. Dificilmente você irá conseguir isto 3 vezes seguidas...).

Ordene agora o seguinte:

```
PRINT 100*RND
```

ora, o que você espera obter? Novamente um número "chutado" só que, desta vez, entre 0 e 100, não mais entre 0 e 1.

Se tirarmos a parte INTEIRA deste número, poderemos usá-lo para jogar na LOTO. Para isto, existe a função INT (tecla R):

```
PRINT INT(100*RND)
```

Assim, obtém-se um número entre 0 e 99.

Faça isto 5 vezes e você terá 5 números para jogar na LOTO essa semana. Vamos então agora "programar" o computador para que ele "execute" essas ordens? Coloque então o seguinte programa:

```
10 RAND
20 PRINT "LOTO"
30 PRINT
40 PRINT INT (RND*100)
50 PRINT INT (RND*100)
60 PRINT INT (RND*100)
70 PRINT INT (RND*100)
80 PRINT INT (RND*100)
```

execute-o diversas vezes (RUN e NEW LINE). Cada vez em que ele for executado, ele pega as próximas 5 cartas com números entre 0 e 1, multiplica-os por 100 e "despreza" as casas depois da vírgula. Assim você terá 5 números INTEIROS entre 0 e 99.

Observe que tivemos que repetir 5 vezes a mesma instrução; imagine se precisássemos repeti-la 100 vezes. Não seria muito agradável ter que fazer isto. Felizmente, existe duas palavra chaves no TK que permitem ordenar ao computador que ele execute diversas vezes uma (ou mais) instruções. Para isto, imagine que alguém peça para você repetir 5 vezes uma dada palavra. Para fazer isto, cada vez que você diz a palavra, você deve estar "contando" em algum lugar da sua cabeça ("memória") o número de vezes que você a repetiu.

Da mesma forma, se desejarmos que o computador repita uma dada instrução, precisamos dizer em que variável da memória ele pode CONTAR o número de vezes em que ele a repetiu. A estrutura deve ser a seguinte:


```
┌►FOR variável - nº inicial TO nº final
│ Instruções a serem repetidas
└─NEXT variável
```

As palavras-chave FOR e NEXT estão nas teclas F e N, respectivamente. O TO está na tecla 4 (não adianta escrever as teclas T e O; deve ser SHIFT 4); assim, o programa anterior poderia ser escrito da seguinte maneira:

```
10 RAND
20 PRINT "LOTO"
30 PRINT
40 FOR I=1 TO 5
50 PRINT INT (RND*100)
60 NEXT I
```

Brincando com Print

Note que usamos a variável I para fazer com que ele contasse de 1 até 5. Lembre-se sempre: para repetir uma ou mais instruções elas devem estar ENTRE um FOR e um NEXT. A este tipo de estrutura chamamos LOOP de repetição. Experimente, então, executar o programa. Que modificação você deverá fazer se quisesse obter 10 números e não 5?

Neste ponto, várias observações são necessárias:

a) ao terminar um loop de FOR-NEXT, a variável contadora estará com um valor de uma unidade maior do que o número de repetições indicado pela palavra-chave FOR. De fato, experimente fazer agora o seguinte comando direto:

```
PRINT I
```

b) a variável contadora, ao contrário das variáveis usadas com a instrução LET deve ter somente uma letra. Com a instrução LET a variável pode ter quantos caracteres quisermos, inclusive números, desde que o primeiro caractere seja uma letra obrigatoriamente:

c) a função INT fornece como resultado o número inteiro de valor imediatamente inferior ao seu argumento; assim temos:

```
INT 5.76=5
INT -5.76=-6
```

d) numa expressão aritmética, todas as funções (cursor K) tem prioridade máxima. Assim, por exemplo, INT 3.4*X é diferente de INT (3.4*X). No primeiro

caso, sendo a função prioritária com relação as demais operações, teremos:

```
INT 3.4*X=3*X
```

Agora, retire as linhas 10, 20 e 30 do programa e substitua as linhas 40 e 50 de tal maneira que seu programa fique:

```
40 FOR I=1 TO 50
50 PRINT I
60 NEXT I
```

Execute o programa. Veja, você ordenou que ele escrevesse a própria variável onde está contando. No entanto, como ele só tem 22 linhas disponíveis, irão aparecer na tela os números de 1 a 22. Mas nós queríamos que ele contasse até 50. Ora, basta mandá-lo continuar. Pressione:

```
CONT
```

e veja o que acontece (este comando encontra-se na tecla C). Repita novamente (quando o programa parar) até obter os 50 números.

Vejamos agora como executar o programa anterior sem que ele pare. Note que os numeros são "impressos" de cima para baixo, parando ao chegar no fim da tela. Existe um comando que faz com que os números sejam escritos de baixo para cima sem interromper o programa. Este comando é a palavra-chave SCROLL (tecla B) e ao ser colocado antes do PRINT, ele simplesmente "rola" a tela para "cima". Assim acrescente a linha:

```
45 SCROLL
```

e execute o programa. Veja o efeito. Perceba que para cada PRINT que colocarmos é necessário um SCROLL. Experimente acrescentar mais duas linhas ao programa de maneira que ele fique:

```
40 FOR I=1 TO 50
45 SCROLL
50 PRINT I
53 SCROLL
56 PRINT "PAZ"
60 NEXT I
```

Se você não colocar o segundo SCROLL, o programa não funcionará.

Execute, então, o programa. Se você desejar "pará-lo" antes que ele termine (não vale desligar o computador) basta apertar a tecla BREAK (a mesma do SPACE). Para continuar, é só apertar CONT; note que este último sempre limpa a tela antes de mandar continuar o programa.

Em capítulos futuros veremos, com maiores detalhes e mais exemplos,a utilização do comando SCROLL.

Neste ponto, podemos falar um pouco mais sobre os números, que aparecem no canto inferior esquerdo da tela quando o programa pára. O número à esquerda indica o "porquê" do programa ter parado; o número à direita, a linha onde o programa parou. Assim, o código 0 indica tudo OK; se você deixar o programa acima terminar "sozinho" você obterá o código:

```
0/60
```

(tudo OK; o programa parou na linha 60).

O código D indica que o Programa parou devido a um BREAK; se você pára o programa assim, irá obter:

D/número que depende do instante em que você apertou a tecla.

Estes códigos aparecem no fim do manual do TK. No decorrer do livro procuraremos analisá-los à medida que aparecerem.

Vamos, agora, relembrar o efeito do ponto e vírgula (;). Retire as linhas 45 e 53 e, a seguir, modifique a linha 50 para:

```
50 PRINT I;
```

e execute o programa.

Acrescente também um ponto e vírgula (;) no final da linha 56:

```
56 PRINT "PAZ";
```

Lembre-se bem do efeito do "ponto e vírgula". Ele informa ao computador que o próximo PRINT deverá ser feito ao lado da última coisa que foi impressa.

Vamos melhorar um pouco a tela, modificando as linhas 50 e 56 de tal maneira que o programa ficará:

```
40 FOR I=1 TO 50
50 PRINT I;" ";
56 PRINT "PAZ ";
60 NEXT I
```

Lembre-se de que, para obter o espaço em branco, deve-se apertar a tecla

SPACE. Após executar o programa apague-o (usando NEW). Vamos introduzir uma nova função: TAB (letra P) que, analogamente a uma máquina de escrever, faz uma tabulação, ou seja, faz com que a impressão comece na coluna que desejarmos; experimente fazer:

```
10 FOR I=1 TO 50
20 PRINT TAB 10;I
30 NEXT I
```

Note que os números são escritos a partir da coluna 10. O que aconteceria se modificassemos a linha 20 para:

```
20 PRINT TAB I;I
```

Faça esta modificação e veja o que ocorre. Você está usando uma variável para fazer a tabulação. Acrescente, agora, a linha 15:

```
15 SCROLL
```

e note o efeito. Quando o argumento do *TAB* for maior que 31, o computador "dá a volta" para recomeçar a escrever no início da tela (para perceber isto no primeiro programa você deveria ter feito dois CONTs). Para observar bem o efeito do *TAB* e do *SCROLL* altere novamente a linha 20:

```
20 PRINT TAB I;I;TAB (2*I);"X"
```

Execute o programa, primeiro com a linha 15 SCROLL e, a seguir, retirando-a. (Neste caso, não esqueça do CONT).

Verifique que se o argumento da função TAB não for inteiro, o computador automaticamente calcula a parte inteira para fazer a impressão na coluna certa. O mesmo é válido para a função AT. Vejamos agora o efeito que a vírgula (,) causa na impressão. Modifique a linha 20 de tal maneira a termos o programa:

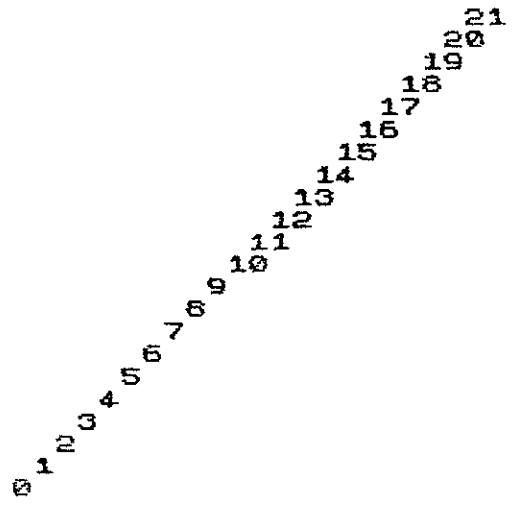
```
10 FOR I=1 TO 50
20 PRINT I,
30 NEXT I
```

Note que a vírgula desloca a próxima posição de PRINT de 1/2 tela. Experimente então modificar a linha 20 para:

```
20 PRINT I,
```


Tente repetir o processo para 3,4 e 5 vírgulas e repare no efeito.

Para finalizar as "palavras novas" deste capítulo, vamos supor que quiséssemos uma "tela" da seguinte maneira:



Isto pode ser feito de maneira análoga a que fizemos agora há pouco para escrever na outra diagonal, se o computador contar de 21 para 0, ou seja, em contagem regressiva. Para isto faça:

```
10 FOR I=21 TO 0 STEP -1
20 PRINT TAB I;I
30 NEXT I
```

A palavra STEP está na tecla E, e ela significa de "quanto em quanto" o computador deve contar. Note que, ao ser omitida, como fizemos até agora, o computador assume STEP igual a 1, ou seja, conta de 1 em 1. No programa que acabamos de mostrar, vai de -1 em -1; não estranhe, o STEP pode ter qualquer valor. Experimente mudá-lo para -2.

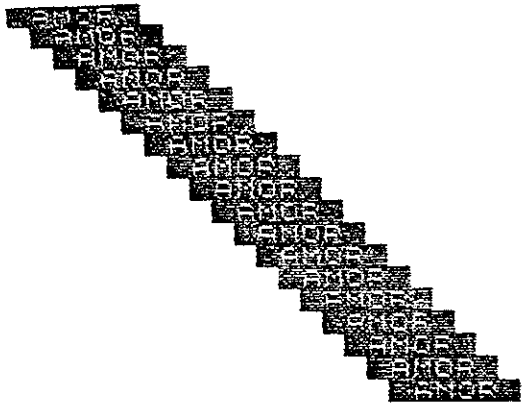
Finalizando, vamos apresentar um pequeno programa para recordar o que vimos até aqui, inclusive a função AT:

```
10 RAND
20 LET N=10
30 FOR I=1 TO N
40 LET A=INT (RND*22)
50 LET B=INT (RND*32)
60 PRINT AT A,B;I
70 NEXT I
```

Repare que a linha 40 "gera" um número aleatório entre 0 e 21 e a linha 50 um número entre 0 e 31, que correspondem respectivamente às linhas e colunas da tela. Assim iremos escrever os números de 1 a 10 em lugares "aleatórios" da tela. Experimente a seguir substituir a linha 20 por:

```
20 LET N=20
```

E execute novamente o programa.
EXERCÍCIOS:
1) Faça um programa que produza o efeito diagonal começando à direita em cima (como no penúltimo programa) SEM utilizar STEP.
2) Faça um programa que escreva várias vezes a palavra AMOR em vídeo reverso (que tenha no começo e no fim um caracter de SPACE também em vídeo reverso) em diagonal, começando no canto superior esquerdo.



3) Faça um programa que gere 13 números aleatórios entre 1 e 3 e apresente a seguinte saída:

```
LOTERIA
3
1
3
3
1
3
2
1
1
3
1
1
2
```

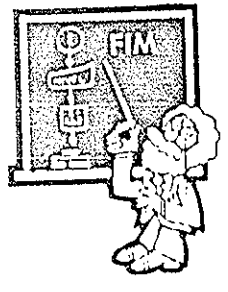

a seguir, implemente a saída para que ela fique assim:

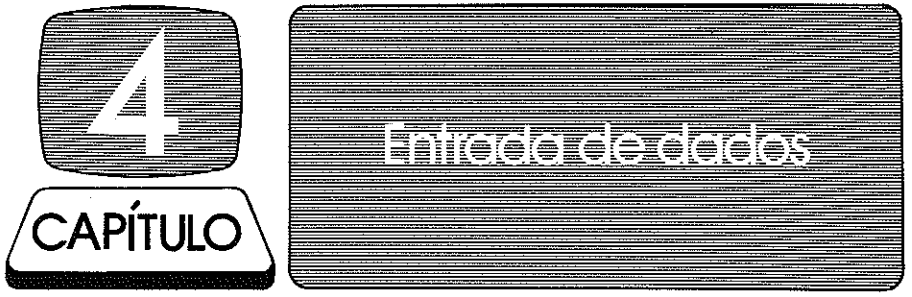
```
LOTERIA
JOGO  1     COLUNA  1
JOGO  2     COLUNA  1
JOGO  3     COLUNA  1
JOGO  4     COLUNA  1
JOGO  5     COLUNA  2
JOGO  6     COLUNA  1
JOGO  7     COLUNA  2
JOGO  8     COLUNA  3
JOGO  9     COLUNA  2
JOGO  10    COLUNA  3
JOGO  11    COLUNA  1
JOGO  12    COLUNA  3
JOGO  13    COLUNA  2
```

Note que a palavra COLUNA não deve aparecer deslocada nos jogos 10, 11, 12 e 13. Finalmente, estude um método para que a saída na tela seja:

```
LOTERIA     1     X     2
JOGO  1     X
JOGO  2     X
JOGO  3     X
JOGO  4                 X
JOGO  5     X
JOGO  6           X
JOGO  7     X
JOGO  8     X
JOGO  9                 X
JOGO  10    X
JOGO  11          X
JOGO  12          X
JOGO  13    X
```

4) Faça um programa que preencha completamente a tela com asteriscos *, a partir da posição 10,10





A Palavra-chave INPUT — Mensagens para o usuário

Até agora, aprendemos a programar o computador para que ele executasse várias ordens seguidas. Entretanto, durante a execução, ele só escrevia coisas na tela, fazia cálculos ou "buscava" coisas em sua memória; em outras palavras, ele apenas "falava" mas, sem "escutar". É como, por exemplo, dizer a uma pessoa: "fale os 10 primeiros números pares." Durante a "execução" da ordem, ela não necessitará nos ouvir, apenas irá dizer os números. Agora experimente dizer a essa pessoa: "eu quero que você some dois números". Ora, o que ela irá fazer? Ela terá que perguntar quais são os números. Analogamente, no computador, existe uma instrução que faz com que, durante a execução, ele pare e espere que você coloque um número através do teclado. Esta instrução está na tecla I: INPUT. Vamos então fazer com que o computador nos pergunte por dois números e a seguir os some:

```
10 INPUT A
20 INPUT B
30 PRINT A+B
```

Execute o programa. A tela fica em branco apenas com o cursor L;não entre em pânico. Ele está obedecendo suas ordens. A primeira instrução diz para ele parar e esperar que alguém digite um número para a seguir colocá-lo na variável A na memória. Coloque então o número 50 e a seguir NEW LINE. Novamente a tela fica em branco; ora, ele está agora esperando o segundo valor.Digite 20 e NEW LINE;finalmente obteremos a soma...Note entretanto que a tela em branco é um inconveniente que pode confundir bastante, principalmente se o programa for razoavelmente complicado. Portanto, é útil escrever algumas"mensagens", principalmente se o seu programa irá ser usado por alguma outra pessoa. Essas mensagens não interferem na lógica do programa, apenas ajudam a sua interpretação durante a execução do mesmo. Experimente então o seguinte:


```
 4 PRINT "EU VOU SOMAR 2 NUMER
OS"
 8 PRINT "QUAL O PRIMEIRO ?"
10 INPUT A
18 PRINT "QUAL O SEGUNDO ?"
20 INPUT B
25 PRINT "A SOMA E" ";
30 PRINT A+B
```

Execute o programa. Note que as linhas 4, 8, 18 e 25 são apenas mensagens. Acrescente então as seguintes linhas:

```
6 PRINT
12 PRINT A
14 PRINT
21 PRINT B
22 PRINT
```

Supondo, por exemplo, que você entre com os números 60 e 40 teremos na tela:

```
EU VOU SOMAR 2 NUMEROS

QUAL O PRIMEIRO ?
60

QUAL O SEGUNDO ?
40

A SOMA E" 100
```

Como você faria para os números 60 e 40 aparecerem não em baixo da pergunta mas *imediatamente* ao lado da mesma? E para que eles aparecessem na mesma linha da pergunta mas na *metade* da tela? (Lembre-se do efeito do ponto e vírgula, e da vírgula ; basta colocá-los convenientemente nas instruções 8 e 18).

Tenha então sempre em mente que, ao programar um computador podemos dizer que três cérebros estão em jogo: o seu, o do computador e o da pessoa que irá buscar o programa. É por isto que as "mensagens" são fundamentais.

CLS e PAUSE — Precisão Numérica do TK

Vamos agora apresentar outra palavra-chave. Trata-se do CLS (tecla V). Acrescente a linha:

```
24 CLS
```

e execute o programa.

O que acontece? Esta instrução APAGA tudo o que havia sido escrito na tela até então. Note entretanto que fica difícil ver o efeito da instrução PRINT B pois apagamos a tela logo em seguida. Para evitar isto, seria necessário uma "espera" ou pausa antes que ele executasse essa instrução. Acrescente então a linha:

```
23 PAUSE 120          (LETRA M)
```

e veja o que ocorre. Esta instrução acrescenta uma PAUSA ao programa onde para casa SEGUNDO de espera devemos adicionar aproximadamente 60. No caso, a pausa é de 2s; se quizessemos uma pausa de 3.5s deveríamos colocar PAUSE 210.

Note que, durante um INPUT, quando o computador está "parado" esperando um número, não adianta fazer BREAK para parar o programa, pois ele está parado; assim, se você quer interrompê-lo, deverá pressionar STOP (SHIFT e tecla A) e a seguir NEW LINE.

Cabem aqui algumas informações quanto à precisão do computador; ele só reconhece 8 dígitos diferentes de zero. De fato, usando o programa; anterior, tente somar 111111111 com 0; você verá que é apresentado um resultado ERRADO pois estamos acima da precisão para este tipo de NOTAÇÃO de números. Note, que se o resultado tiver zeros no final, ele é capaz de representar até a 13ª casa; some 999999999 com 1; você obterá 1E + 13 que significa 1×10^{13}; assim o computador é capaz de interpretar números com mais de 8 dígitos desde que usemos esta notação. O mesmo é válido para números muito pequenos. O maior número em valor absoluto que o computador TK aceita é:

1.7014118E + 38

e o menor em valor absoluto (o mais próximo de zero)

2.9387359E - 39

ABS e GOTO; exemplo prático

Vamos fazer um programa que utilize as instruções aqui apresentadas, e mais uma função que está na tecla G (ABS) que serve para calcular o valor absoluto de um número. Para entendê-la, experimente antes fazer este pequeno programa:

```
10 FOR I=10 TO -10 STEP -1
20 PRINT TAB (ABS I);I
30 NEXT I
```


Que tal o efeito?

Feito isto, iremos então fazer um programa para "treinar a tabuada", que pergunte para você o resultado de uma operação de multiplicação usando números chutados por ele, escrevendo a sua resposta, a do computador, e a diferença entre elas, em valor absoluto. Vamos fazer com que ele pergunte também qual o valor máximo dos números que ele deve multiplicar:

```
 10 RAND
 20 PRINT "EU VOU PERGUNTAR A V
OCE QUANTO  VALE O PRODUTO DE 2
NUMEROS"
 30 PRINT "QUAL O NUMERO MAXIMO
 QUE EU     POSSO USAR ?"
 40 INPUT MAX
 50 CLS
 60 LET A=INT (2*MAX*RND+1) -MAX
 70 LET B=INT (2*MAX*RND+1) -MAX
 80 PRINT A;"X";B;"=?"
 90 INPUT P
100 PRINT
110 PRINT "SUA RESPOSTA ";P
120 PRINT "RESPOSTA CORRETA ";A
*B
130 PRINT
140 PRINT "DIFERENCA ";ABS (P-A
*B)
150 PAUSE 180
160 CLS
170 GOTO 20
```

Note que, na linha 170, foi usada uma palavra-chave, GOTO (tecla G), que significa "vá para" dada instrução (no caso 20); seu efeito é análogo ao RUN, só que ele não executa um CLEAR. De fato, pare o programa (dando BREAK ou STOP conforme o caso) e, execute-o novamente; só que, ao invés de fazer RUN e NEW LINE, faça GOTO 10 e NEW LINE.

Preste atenção na estrutura utilizada nas linhas 60 e 70 para gerar números positivos ou negativos com valor absoluto menor do que o valor máximo.

EXERCÍCIOS:

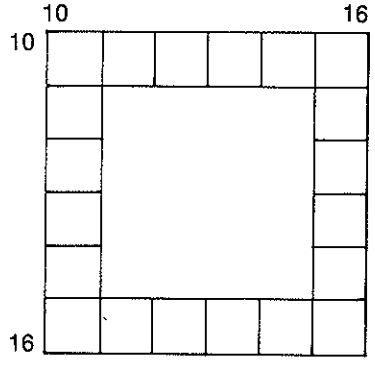
1) Faça um programa para imprimir na tela os primeiros 44 pares (0 até 86), sendo que todos os números caibam na tela e apareçam de 1 em 1 segundo.

2) Elabore um programa que faça o produto de três números que faça todas as perguntas necessárias ("mensagens") e, no fim, a tela mostre algo do tipo:

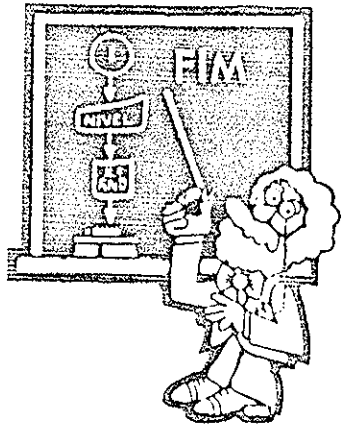
```
A=2

B=6

C=5

PRODUTO=60
```

3) Complemente o programa da LOTO para que ele pergunte a você quantos números você gostaria de jogar e, a seguir, ele "gere" os números na quantidade desejada.

4) Faça um programa que desenhe na tela um quadrado usando quaisquer símbolo gráfico da seguinte maneira:



A seguir deve perguntar a você até que números ele deve contar e, feito isto, ele começa a contar colocando os número no centro do quadrado (posição 13, 13), sendo que cada número impresso substitui o anterior.

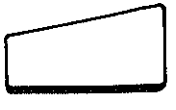


Tomada de decisões pelo computador

Neste capítulo, iremos introduzir um novo tipo de raciocínio usado para elaborar programas: trata-se de um raciocínio gráfico que facilita a visualização de um programa e cuja utilidade torna-se evidente, quanto mais complexo for o programa a ser utilizado. Ele é denominado *fluxuograma* ou *diagrama de blocos* e baseia-se no fato de que podemos associar, a cada idéia básica usada em programação, um símbolo, sendo que os vários símbolos são interligados por linhas orientadas, que indicam o fluxo de execução do programa. Outra vantagem deste método é a sua universalidade; de fato, como as idéias básicas de qualquer linguagem são as mesmas, uma vez feito um programa em diagrama de blocos, ele poderá ser facilmente "traduzido" para BASIC, FORTRAN, ASSEMBLY, COBOL, ou qualquer outra linguagem desde que ela não seja estruturada (o significado deste termo não pode ser explicado num livro introdutório como este).

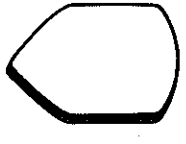
Até o presente momento, as idéias básicas vistas por nós foram:

a. Entrada de dados por teclado:



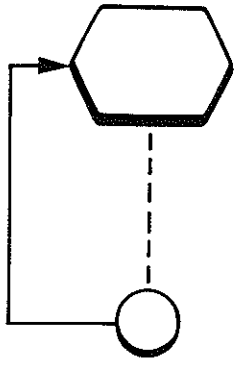
(BASIC = INPUT)

b. Saída de dados para a tela:



(BASIC = PRINT)

c. Loops:



(BASIC = FOR/NEXT)

d. Operações e funções:

(BASIC = LET/CLS/RAND/ +/-/ /

e. Início de programa:

f. Fim de programa:

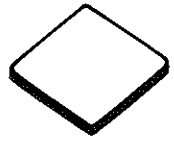
g. Continuação do fluxograma, quando não há espaço físico:

* qualquer símbolo sem ser I, F ou R

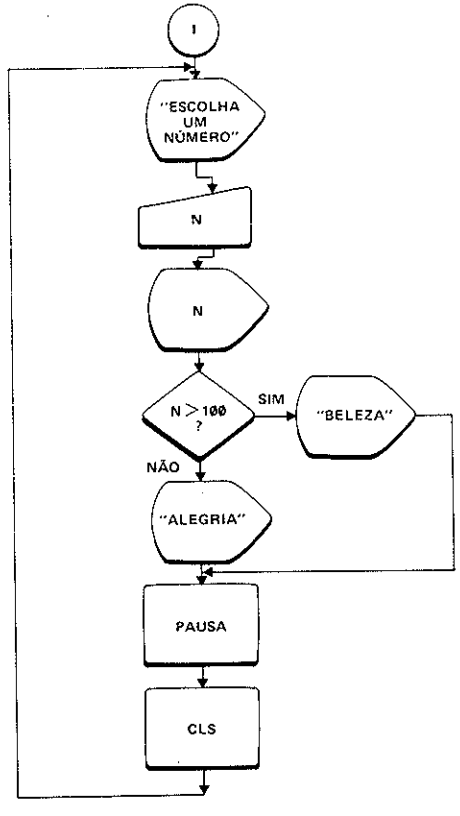
A palavra-chave IF

Vamos apresentar agora um outro conceito, talvez o mais importante em programação: a tomada de decisões por parte do computador.

Esta é uma ordem que torna o computador capaz de decidir, baseado numa condição, especificada por você, qual "caminho" seguir na execução de um programa. Esta característica é que faz com que o computador seja chamado de "máquina inteligente". O símbolo utilizado para tomada de decisões é :



Para exemplificar, façamos um programa que "pergunte" um número, coloque-o na variável N e escreva na tela BELEZA, se o número for maior do que 100 e ALEGRIA em caso contrário. Este seria o fluxograma:



Note que a idéia de tomada de decisão, ao contrário das demais, implica em uma "entrada" e pelo menos "duas saídas" (todas as outras têm apenas uma entrada e uma saída). Surge assim uma pequena dificuldade ao tentar passar este programa para o BASIC pois as instruções são escritas sequencialmente ou seja, numa única "dimensão". A instrução de decisão tem a seguinte forma:

IF condição THEN faça algo

(Caso contrário, prossiga para a próxima instrução.)

A palavra-chave IF está na tecla U e o THEN na tela 3. Note que esta estrutura é a única que permite o aparecimento de *duas* palavras-chaves numa única instrução. De fato, após o THEN, o cursor volta a indicar **K** .

Vamos então ao nosso programa:

```
 10 PRINT "ESCOLHA UM NUMERO";
 20 INPUT N
 30 PRINT N
 40 PRINT
 50 IF N>100 THEN GOTO 100
 60 PRINT "ALEGRIA"
 70 PAUSE 150
 80 CLS
 90 GOTO 10
100 PRINT "BELEZA"
110 GOTO 70
```

Na linha 50 o TK toma a decisão: se N > 100 , ele vai para a linha 100 ; caso contrário, prossegue para a linha 60. (O símbolo > está na tecla M).

Perceba como é mais simples visualizar o programa em diagrama de blocos. Note que o GOTO não tem símbolo especial: ele corresponde a uma linha orientada. Vale a pena salientar também que não é necessário incluir detalhes nos fluxuogramas: por exemplo, no programa anterior, a instrução de pular linha não foi incluída. Apenas as idéias principais devem ser "desenhadas"; isto facilita a visualização.

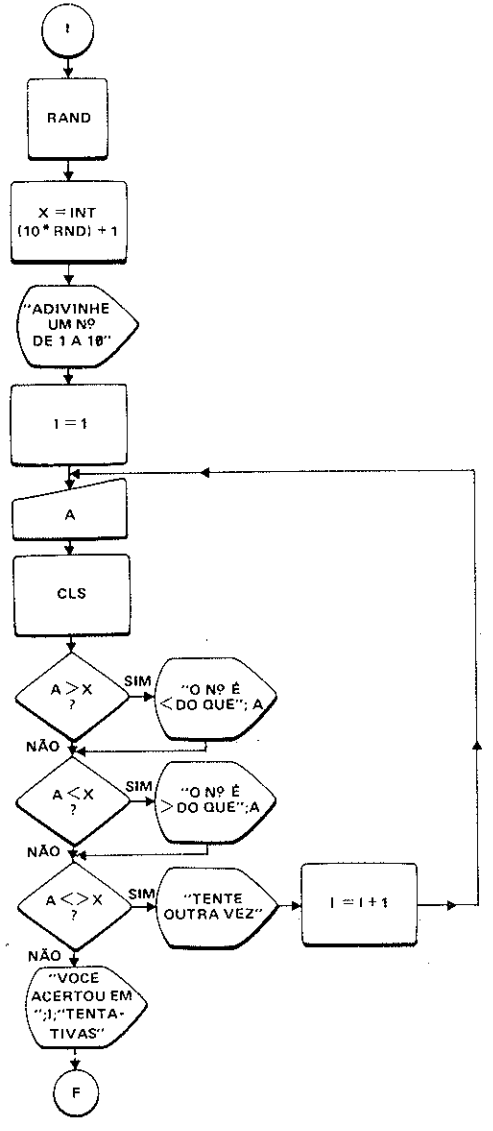
As velocidades do TK

Antes de continuar com mais exemplos de utilização do fluxograma vamos apresentar mais duas instruções. O TK tem duas velocidades de trabalho, uma lenta (SLOW) e outra rápida (FAST). Quando ele opera em SLOW, ele se preocupa em executar o programa e, ao mesmo tempo, mostrar os resultados na tela; em FAST, ele "esquece" a tela e os resultados só são apresentados durante uma PAUSA ou no final do programa. Naturalmente, cada uma tem suas vantagens e tudo depende de cada caso. Quando o TK é ligado, ele automaticamente opera em SLOW. No entanto, para digitar um programa, é mais conveniente usar a velocidade mais rápida; de fato, experimente digitar:

```
LET X=12345678901234567890
```


A seguir, digite FAST (Shift F) e NEW LINE e repita a operação acima. (Obs: para entender bem estes comandos, execute os programas dados do capítulo 2 em SLOW e em FAST).

Vamos então fazer um programa que "chuta" um número; você terá que adivinhá-lo no menor número de tentativas possível.



(Obs: O símbolo <> significa ≠ , ou *diferente*).
O que em BASIC ficaria:

```
  10 RAND
  15 LET X=INT (10*RND)+1
  20 PRINT "ADIVINHE UM NUMERO D
E 1 A 10 QUEEU VOU CHUTAR"
  30 LET I=1
  40 INPUT A
  50 CLS
  60 IF A>X THEN PRINT "O NUMERO
 QUE EU CHUTEI E MENOR  DO QUE "
;A
  70 IF A<X THEN PRINT "O NUMERO
 QUE EU CHUTEI E MAIOR  DO QUE "
;A
  80 IF A<>X THEN GOTO 100
  90 GOTO 130
 100 PRINT "TENTE OUTRA VEZ"
 110 LET I=I+1
 120 GOTO 40
 130 PRINT "VOCE ACERTOU EM ";I;
" TENTATIVAS"
```

Execute o programa. Perceba o uso da variável I como "contador" de tentativas. O símbolo <> está na tecla T e < está na tecla N. Este programa poderia ser escrito mais "elegantemente" eliminando a linha 90 e substituindo a linha 80 por:

```
80 IF A=X THEN GOTO 130
```

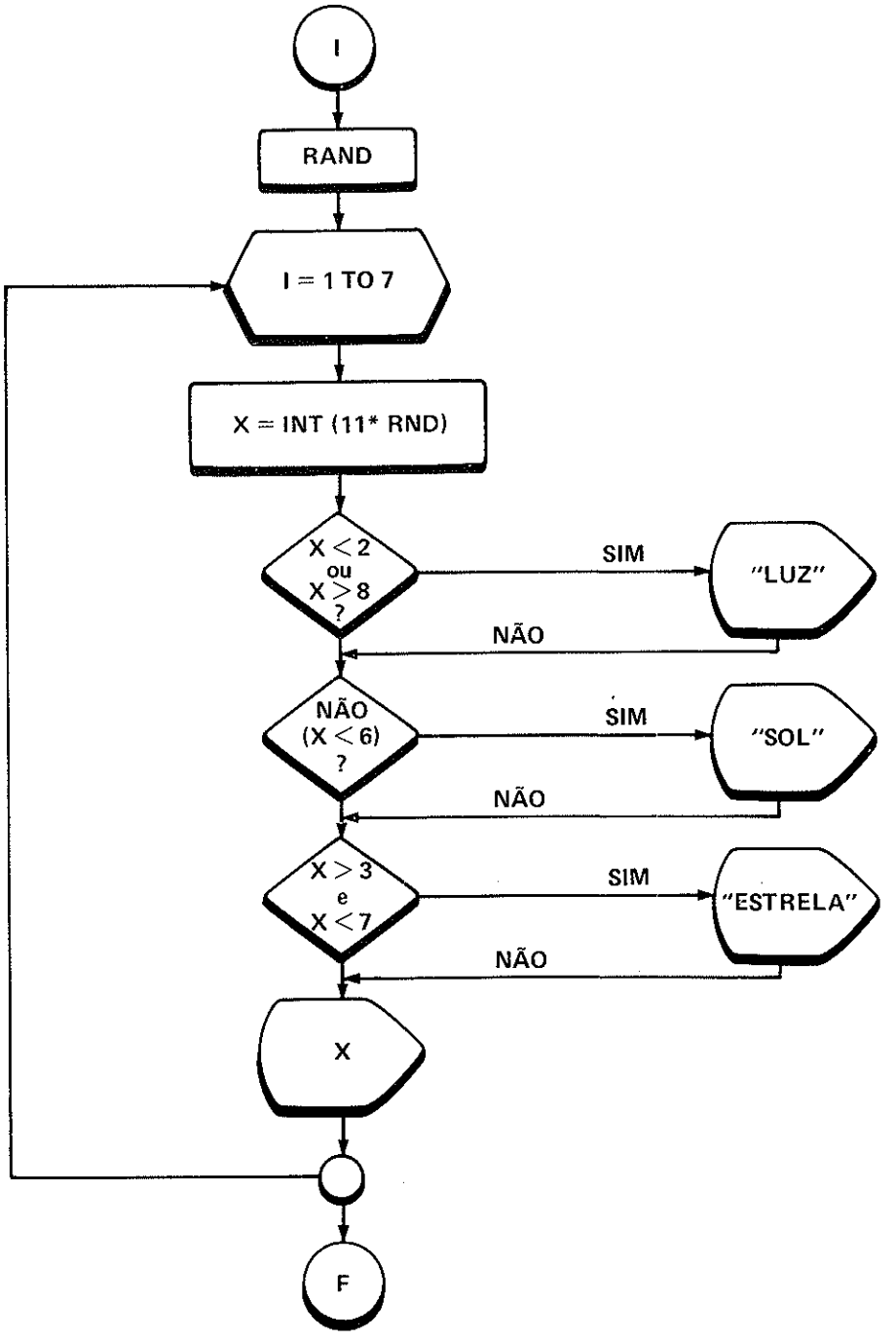
Um pouco de lógica: AND, OR, NOT

Existem no TK três "palavras" que podem ser muito úteis quando associadas ao IF; são elas:
AND (tecla 2)
OR (tecla W)
NOT (tecla N FUNCTION)
AND significa "e"; NOT significa "não", enquanto OR significa"ou".

Suponha que no programa que imprimia "BELEZA" ou "ALEGRIA" mudássemos as condições: se o número não estiver entre 30 e 70 imprima "BELEZA". Para isso, bastaria substituir a linha 50:

```
50 IF N>30 AND N<70 THEN GOTO 100
```

Para ilustrar as outras duas, façamos um pequeno programa:



Note que temos em jogo o seguinte conjunto de números possíveis:

(0,1,2,3,4,5,6,7,8,9,10)

Para x<2 ou x>8, teremos - (0,1,9,10)
Para NÃO (x<6) ≡ x>6 - (6,7,8,9,10)
Para x>3 e x<7 - (4,5,6,)
Em BASIC teríamos:

```
  5 SLOW
 10 RAND
 20 FOR I=1 TO 7
 30 LET X=INT (11*RND)
 40 IF X<2 OR X>8 THEN PRINT "L
UZ"
 50 IF NOT X<6 THEN PRINT "SOL"
 60 IF X>3 AND X<7 THEN PRINT "
ESTRELA"
 70 PRINT X
 80 NEXT I
 85 FAST
```

Estas palavras aliadas ao IF serão discutidas novamente em capítulos futuros.

Note que temos os símbolos, menor ou igual < = (tecla R), e maior ou igual > = (tecla Y), que também podem ser utilizadas com a instrução IF.

Vamos agora concluir todos os símbolos "vermelhos" (ou amarelos) do TK, com exceção do $ que será visto futuramente; faltam apenas:

```
:
?
£
STOP (TECLA A)
LPRINT (TECLA S)
LLIST (TECLA G)
```

Os primeiros três não têm nenhuma função especial e servem apenas para participar de "mensagens", que devem estar sempre entre aspas. Pode-se acrescentar a eles o ponto (.) que está na mesma tecla que a vírgula (,). Já o STOP serve para "quebrar" a execução de um programa. Um programa pode constar de várias partes cada uma terminada por STOP. Ao chegar ao STOP o programa pára e ele pode ser reiniciado a partir da instrução seguinte, usando CONT (veja capítulo 3).

Por exemplo:


```
  5 SLOW
 10 PRINT TAB 5;"BEATLES"
 20 STOP
 30 PRINT TAB 5;"IS LOVE"
 40 STOP
 50 PRINT TAB 5;"FOREVER"
 60 STOP
 70 PRINT
 80 LET I=2
 90 PRINT TAB 7;I
100 IF I=256 THEN STOP
110 LET I=I*I
120 GOTO 90
```

Digite RUN e NEW LINE; a seguir, você terá que digitar CONT e NEW LINE nas primeiras três vezes que o programa parar. O que acontece se você fizer CONT e NEW LINE quando o programa parar pela quarta vez?

Note que, ao parar por STOP, o TK imprime a seguinte mensagem no canto esquerdo inferior:

```
9/(LINHA QUE PAROU O PROGRAMA)
```

(assim 9 é código de STOP).
Obs: O STOP já havia sido mencionado suscintamente no capítulo 4 quando falamos sobre o INPUT.

Utilizando a impressora

Quanto ao LPRINT e ao LLIST eles fazem o mesmo que os já conhecidos PRINT e LIST só que ao invés de escrever na tela de TV escrevem na impressora. Ainda para lidar com a impressora, existe a palavra chave COPY (tecla Z), que copia exatamente o que está desenhado ou escrito na tela. (Naturalmente você pode fazer um BREAK durante um COPY se não desejar copiar a tela toda).

Algumas observações são necessárias:

a) O LLIST permite, assim como o LIST, fazer uma listagem a partir de qualquer linha do programa; além disso, ele pode ser "interrompido" por um BREAK.

b) Para programas "menores" que uma tela, é conveniente usar COPY ao invés de LLIST pois ele é mais rápido.

c) Não existe a instrução LPRINT AT (tente executá-la...) mas existe a instrução LPRINT TAB.

d) Com exceção do AT e do SCROLL, o PRINT e o LPRINT são perfeitamente compatíveis com relação ao posicionamento da impressão (TAB, ;); no entanto, se você for utilizar números na faixa de 0.00001 a 0,00 99999999 a instrução não funciona adequadamente, experimente:

```
10 LET X=0.00001
20 LPRINT X
```

Em capítulos futuros, veremos como contornar este problema.

3) O LPRINT tem um símbolo especial para diagrama de blocos.

Exercícios:

1) Faça um programa capaz de calcular o fatorial de um número inteiro positivo. O programa **não** deve aceitar números negativos e/ou fracionários imprimindo uma mensagem de erro.

Define-se fatorial de **n**, como sendo:

n! — n.(n-1). (n-2)...1
Assim
5! — 5.4.3.2.1 — 120

2) Escreva um programa que leia o numero e mostre, na tela, se o número é par ou ímpar. Seguindo o mesmo raciocínio, faça então outro programa que escreva na tela "múltiplo de 6" todas vez que o número for múltiplo de 6 - (Dica: **a** ÷ **b** com resto **c** ; como se calcula **c** ?)

3) Elabore um programa que pergunte a você quantas operações de multiplicação você deseja fazer e com quantos fatores cada uma. A seguir ele "chuta" as várias contas, pergunta a resposta a você e, no final, ele deve imprimir o número de erros, o número de acertos e a sua nota de 0 a 10.

4) Faça um programa que calcule as raízes de uma equação do 2° grau do tipo $ax^2 + bx + c$ dados **a, b e c**. Ele deve imprimir mensagem de erro caso o $\triangle$ seja menor do que zero. Lembre-se que as raízes de uma equação do 2° grau são dadas por:

$$x_1 = \frac{-b+\sqrt{\triangle}}{2a} \qquad x_2 = \frac{-b-\sqrt{\triangle}}{2a}$$

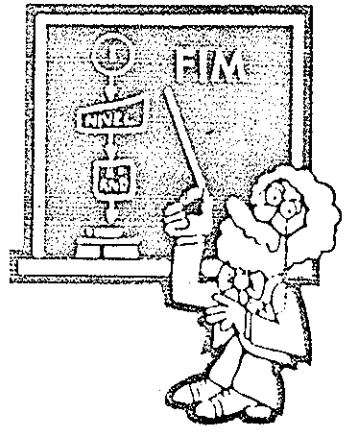
com $\triangle = b^2 - 4ac$
para calcular a raiz quadrada de um número basta elevá-lo à potência 0.5; assim:

25 ** 0.5 = 5

Se você preferir, existe no TK a função SQR (tecla H) que calcula diretamente a raiz quadrada:

```
PRINT SQR 25
```

NOTA: Procure raciocinar sempre em diagrama de blocos daqui para frente, a não ser para os programas mais simples.



6 CAPÍTULO
Grafismos e Efeitos visuais

Neste capítulo, iremos explorar a potencialidade do TK com relação à formatação da tela, ou seja, o posicionamento das saídas de um programa da tela. Vimos até o presente momento que, usando PRINT, é possível escrever na tela valores de variáveis ou mensagens escritas entre aspas ("). O uso de vários PRINTs seguidos faz que cada instrução "escreva" na linha imediatamente após a anterior e este efeito pode ser "cancelado" utilizando o ponto e virgula (;). Para recordar, experimente executar os seguinte programas:

```
10 SLOW
20 FOR I=1 TO 22
30 PRINT "▗▙▄▀▟▌"
40 NEXT I
50 CLS
60 PRINT "▗▙▄▀▟▌";
70 GOTO 60
```

(Note que ao terminar este programa aparece no canto inferior esquerdo da TV, o código 5/60 onde 5 indica que não há mais espaço na tela).

```
10 SLOW
20 PRINT "BEATLES";" ";
30 PRINT "LOVE";" ";
40 GOTO 20
```

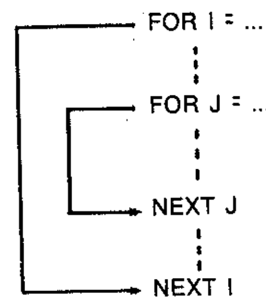
Vimos também o efeito da vírgula que fazia "pular" meia tela, a posição do PRINT, as funções TAB e AT (recordando que a tela tem 24 linhas e 32 colunas) e a SCROLL. Vamos recordar estes conceitos:

```
10 SLOW
20 FOR I=21 TO 0 STEP -1
30 PRINT TAB I;"▞";TAB (I+10);
"▚"
40 NEXT I
50 PRINT AT 10,10;"SDROBILO","
MAMAE"
```


Execute agora o seguinte programa que ilustra claramente o efeito de "rolar" a tela:

```
10 SLOW
20 FOR I=1 TO 22
30 SCROLL
40 FOR J=1 TO 4
50 PRINT "▟▛▙▜";
60 NEXT J
70 NEXT I
80 FOR I=1 TO 22
90 SCROLL
100 NEXT I
110 GOTO 20
```

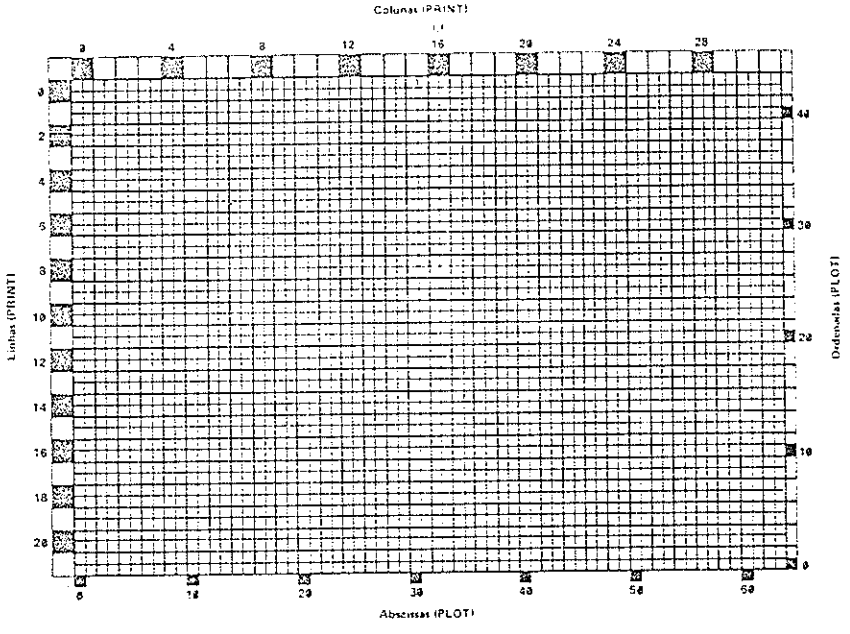
Note que estamos usando um loop dentro de outro loop. Dessa maneira, o loop interno será repetido tantas vezes quantas forem indicadas pelo loop externo. Note que os NEXT devem aparecer em ordem inversa aos FOR, ou seja:



Aumentando a resolução das coordenadas: PLOT e UNPLOT

Vamos apresentar agora a resolução máxima que o TK é capaz de conseguir (sem ajuda de dispositivos especiais); imagine um papel "quadriculado" com eixos horizontal e vertical, e posições numeradas de 0 a 43 (coordenadas) e 0 a 63 (abcissas). Dessa mesma maneira a tela do TK é dividida para permitir o uso de palavra-chave PLOT (tecla Q) que coloca um ponto preto em dada coordenada, cujas dimensões são exatamente um quarto do caractere "█" (SPACE em vídeo reverso). Note entretanto que o ponto 0, 0 para o PLOT correspondente ao canto inferior esquerdo da tela enquanto que a posição 0, 0 para PRINT AT, corresponde ao canto superior esquerdo. Execute o seguinte programa:

```
10 SLOW
20 FOR I=0 TO 21
30 PRINT AT I,I;"█"
40 PLOT I,I
50 NEXT I
```



Temos também a palavra-chave UNPLOT (tecla W) que apaga um ponto que havia sido "plotado" anteriormente. Acrescente as seguintes linhas ao programa anterior, e execute-o novamente.

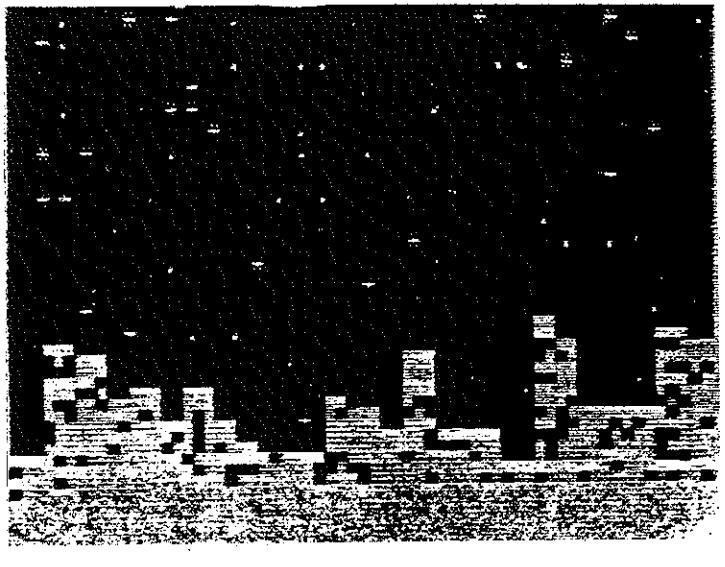
```
42 PRINT AT I,I;" "
44 UNPLOT I,I
```

Experimente então o seguinte programa que, além de mostrar claramente a diferença entre PRINT AT e PLOT, produz um belo efeito "artístico":

```
10 SLOW
20 FOR I=0 TO 21
30 FOR J=0 TO 31
40 PRINT AT I,J;"▒"
50 PLOT I,J
60 NEXT J
70 NEXT I
```


Faremos agora uma pequena aplicação destas funções para desenhar uma "imagem" de São Paulo à noite, sob um céu estrelado.-. sem lua...

```
 10 SLOW
 20 RAND
 30 FOR K=1 TO 88
 40 PRINT "▒";
 50 NEXT K
 60 FOR I=1 TO 22
 70 FOR K=0 TO 31
 80 LET S=1+INT ((9+I*2)*RND)
 90 IF S=2 THEN PRINT AT I-1,K;
"▒"
100 IF S=4 THEN PRINT AT I-1,K;
"▒"
110 NEXT K
120 NEXT I
130 LET X=0
140 LET H=2+INT (14*RND)
150 FOR L=1 TO 2+INT (2*RND)
160 FOR Y=0 TO H
170 UNPLOT X,Y
180 IF Y>0 AND Y<H AND (4*RND)>
3 THEN PLOT X,Y
190 NEXT Y
200 LET X=X+1
210 IF X=64 THEN GOTO 240
220 NEXT L
230 GOTO 140
240 FOR Y=0 TO 63
250 PLOT Y,0
260 NEXT I
```

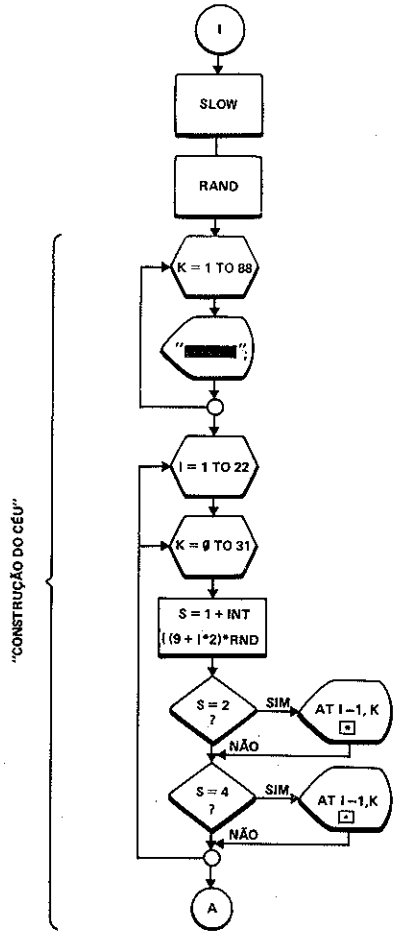


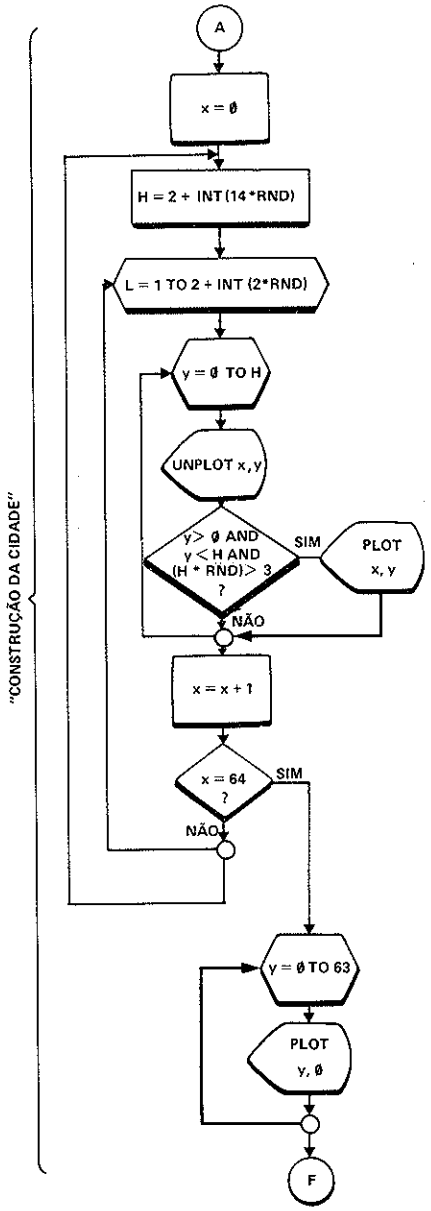
Note como a linha 80 foi construída de tal maneira que a probabilidade de termos estrelas é maior nas linhas mais "altas" da T.V. Estude também o funcionamento da linha 180...

Caso você não tenha paciência de ficar esperando a "construção" da cidade, substitua a linha 10.

```
10 FAST
```

Como você faria para aumentar a quantidade de estrelas visíveis (supondo que seja possível diminuir a poluição...) ? E para aumentar a altura dos prédios? Dê uma olhada no fluxograma do programa, que mostramos a seguir :





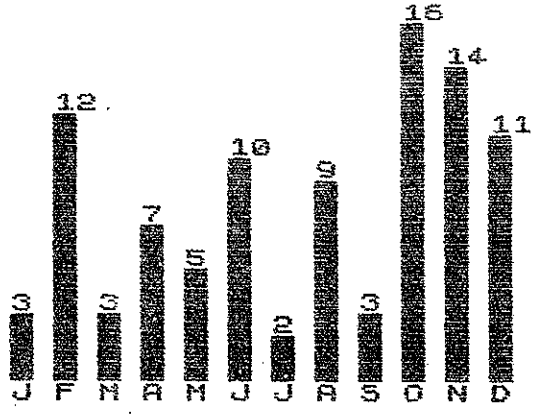
Com PLOT e UNPLOT temos então a máxima resolução gráfica disponível normalmente para o TK (44 x 64 pontos). Entretanto já existem no mercado dispositivos capazes de aumentarem esta resolução até 192 a 256 pontos.

Existem inclusive periféricos que possibilitam a criação de novos caracteres...

Outra aplicação das instruções para formatação da tela é a construção de histogramas (ou gráficos de barras). Por exemplo:

```
  5 RAND
 10 SLOW
 20 PRINT "HISTOGRAMA DAS VENDA
S DE          ABACAXI"
 30 PRINT "EM BILHOES DE TON.
ANO:2001"
 40 PRINT AT 21,4;"J"
 50 PRINT AT 21,6;"F"
 60 PRINT AT 21,8;"M"
 70 PRINT AT 21,10;"A"
 80 PRINT AT 21,12;"M"
 90 PRINT AT 21,14;"J"
100 PRINT AT 21,16;"J"
110 PRINT AT 21,18;"A"
120 PRINT AT 21,20;"S"
130 PRINT AT 21,22;"O"
140 PRINT AT 21,24;"N"
150 PRINT AT 21,26;"D"
160 FOR C=4 TO 26 STEP 2
170 LET X=INT (16*RND)+1
180 FOR L=20 TO 21-X STEP -1
190 PRINT AT L,C;"■"
200 NEXT L
210 PRINT AT L,C;X
220 NEXT C
```

```
HISTOGRAMA DAS VENDAS DE
ABACAXI
EM BILHOES DE TON.  ANO:2001
```



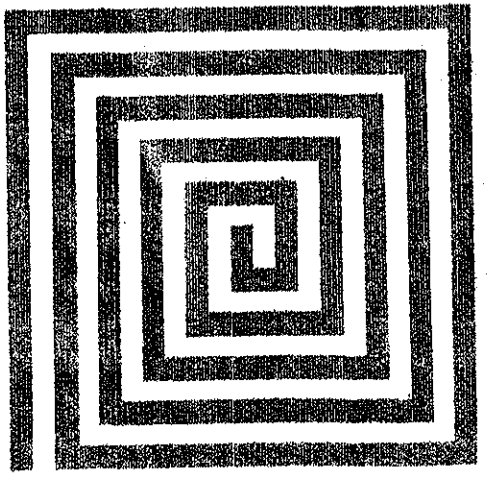
Tente explicar o funcionamento da linha 180...

EXERCÍCIOS

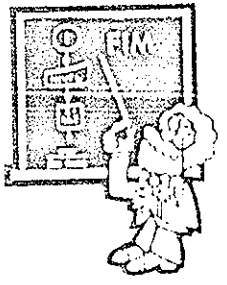
1. Faça um programa que produza o mesmo efeito que o terceiro programa deste capítulo sem usar STEP.

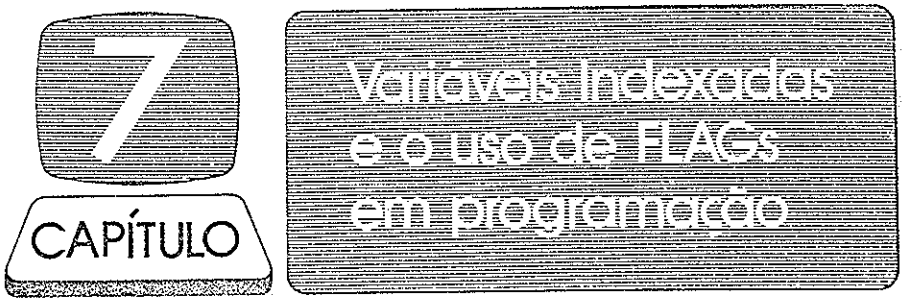
2. Elabore um programa que escreva um caractere "simultaneamente" nas duas diagonais.

3. Elabore um programa que seja capaz de desenhar uma "espiral quadrada"...



Obs.: Procure raciocinar utilizando diagrama de blocos quando necessário.





Colocando tabelas na memória: a palavra-chave DIM

No capítulo 1 introduzimos a palavra-chave LET que associava a uma variável numérica um "lugar" na memória. Entretanto existem ocasiões em que é desejável armazenar vários valores para uma "mesma" variável; por exemplo, para o preço de um dado produto ao longo do ano. Podemos então usar várias vezes uma "mesma" variável, distinguindo seus diferentes valores através de índices entre parênteses; seria como se construíssemos uma tabela na memória à qual denominamos Matriz. Assim, temos a palavra-chave DIM (tecla D) cuja função é reservar espaço na memória para os valores que desejamos colocar, atribuindo inicialmente valor zero a todos eles.

Experimente então o seguinte programa:

```
 10 SLOW
 20 DIM X(5)
 30 FOR I=1 TO 5
 40 PRINT "X(";I;")=";X(I)
 50 NEXT I
 60 PRINT
 70 FOR I=1 TO 5
 80 PRINT "X(";I;")=?";
 90 INPUT X(I)
100 PRINT AT I+5,5;X(I)
110 NEXT I
```

Na instrução 20 reservamos na memória para 5 valores de X colocando inicialmente valor 0 (zero) para os mesmos; a seguir, usando INPUT associamos valores aos vários X e usando um "pequeno truque" na linha 100 escrevemos na tela os números colocados. Note a diferença de significado dos índices; com a palavra-chave DIM ele indica o tamanho máximo da tabela; nas demais instruções, ele indica a posição na tabela. Verifique o que acontece no programa anterior se acrescentarmos a linha:

```
95 DIM X(5)
```

Infelizmente no BASIC TK estas variáveis indexadas tem uma limitação: ao contrário das variáveis normais, seu "nome" pode ter apenas uma letra; assim, por exemplo, o seguinte comando é inválido:

```
DIM PAZ(34)
```

Em contrapartida, é possível fazer matrizes (TABELAS) com mais de uma dimensão (até 255 dimensões). Por exemplo, duas dimensões são facilmente imagináveis se pensarmos em linhas e colunas. Por exemplo:

```
  5 SLOW
 10 DIM A(3,4)
 20 FOR I=1 TO 3
 30 FOR J=1 TO 4
 40 PRINT AT 4,4;"A(";I;",";J;"
)=?"
 50 INPUT A(I,J)
 60 PRINT AT 4,11;A(I,J)
 70 PAUSE 90
 80 CLS
 90 NEXT J
100 NEXT I
110 PRINT TAB 4;"MATRIZ ESCOLHI
DA"
120 PRINT
130 FOR I=1 TO 3
140 FOR J=1 TO 4
150 PRINT AT 2*I+2,4*J;A(I,J)
160 NEXT J
170 NEXT I
```

Repare que, na linha 10, reservamos espaço para uma tabela de 3 linhas e 4 colunas, ou seja, para 3x4 = 12 variáveis com valor inicial zero. A seguir, você deve fornecer os valores para cada variável (INPUT) e finalmente a matriz é colocada na tela. Repare no uso da instrução AT para a formatação da saída na linha 150; com esta formatação, qual o número máximo de algarismos que posso usar para cada valor? É possível aumentá-los? Você saberia "imaginar" uma matriz com 3 ou mais dimensões?

Noção de FLAG; utilização na ordenação de números

Denominamos FLAG a uma *variável auxiliar* cuja função é indicar a ocorrência ou não de determinada condição durante a execução de um programa. Uma utilização prática para este tipo de variável está na ajuda em descobrir erros em programas razoavelmente complicados, principalmente

aqueles com um grande número de instruções IF. Nestes programas, a quantidade de "caminhos" possíveis é muito grande, o que dificulta a análise do mesmo passo a passo. Uma analogia pode ser feita com uma linha de trens bastante intrincada e com muitos desvios .

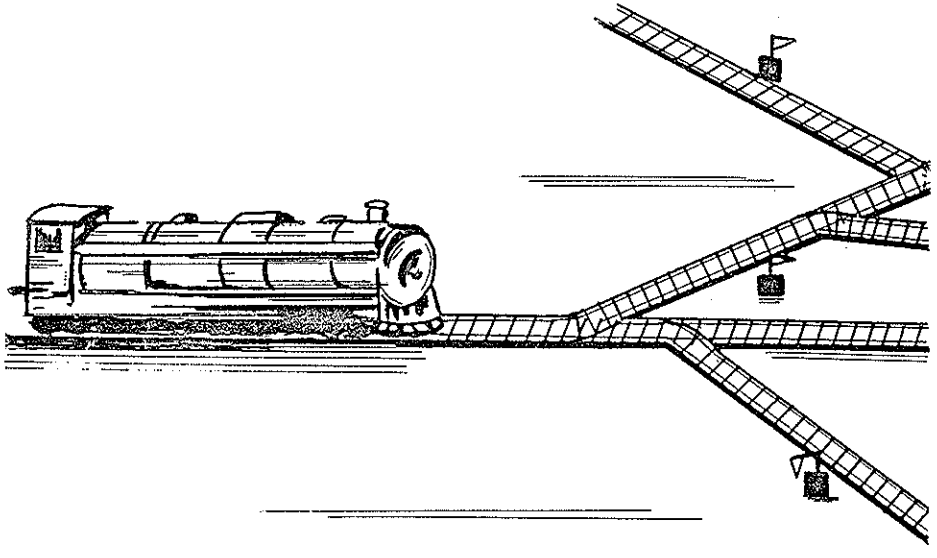
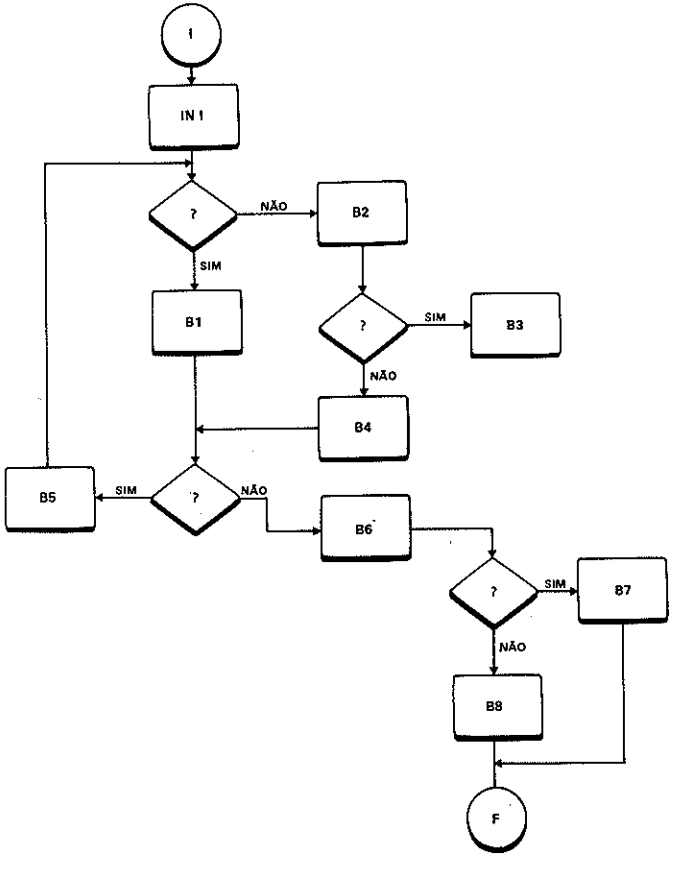


fig. 7.1

Suponha que um trem irá passar por uma dada região e você gostaria de saber qual o caminho percorrido pelo trem, mas sem poder ver o trem passar... Se, para cada desvio, tivermos um mecanismo que levanta uma bandeira ao passar o trem e se inicialmente todas as bandeiras estiverem abaixadas, basta verificar quais estão levantadas após o trem passar para saber o caminho percorrido. Do mesmo modo, ao executar um programa, não podemos ver o "trem passar". Verifique que o pequeno programa com 4 desvios, que mostramos na figura



fluxograma C

Cada retângulo representa um bloco de instruções; no bloco inicial (INI) basta "abaixar" todos os FLAGs que podem ser representados por uma matriz de uma dimensão. Usando a convenção de 0 para FLAG abaixado e 1 para FLAG levantado, é suficiente fazer:

```
DIM F(8)
```

A primeira instrução de cada bloco deve ser um "Levantamento" da FLAG respectiva, assim:

```
Início do bloco 1 ──────────► LET F(1)=1
Início do bloco 2 ──────────► LET F(2)=1
        .                        .
        .                        .
        .                        .
        .                        .
Início do bloco 8 ──────────► LET F(8)=1
```

Após executar o programa, basta verificar qual o "estado" das FLAGs para se ter uma idéia do caminho percorrido:

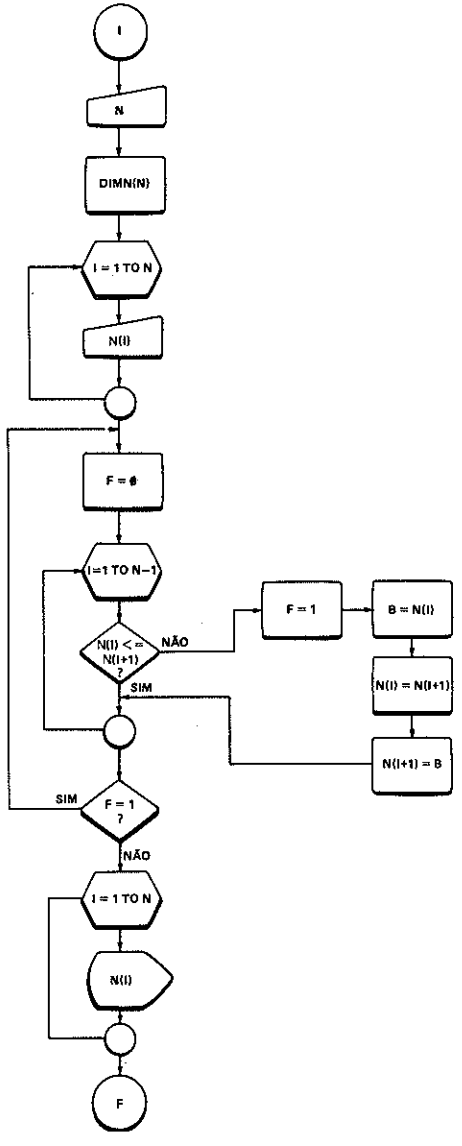
```
1000 FOR I=1 TO 8
1005 SCROLL
1010 PRINT "FLAG ";I;" ";F(I)
1015 NEXT I
```

Naturalmente as FLAGs usadas desse modo não indicam quantas vezes o programa passou por dado caminho. Se isto for necessário, basta modificar a instrução no início de cada bloco:

```
Início do bloco 1 ──────────► LET F(1)=F(1)+1
Início do bloco 2 ──────────► LET F(2)=F(2)+1
        .                        .
        .                        .
        .                        .
        .                        .
Início do bloco 3 ──────────► LET F(8)=F(8)+1
```

Iremos agora utilizar um FLAG para ordenar uma lista de números em ordem crescente. O raciocínio adotado será o seguinte:

Ao findar a lista, testamos o valor do FLAG e, se houver alguma troca, repetimos o processo. Isto é feito até que não seja necessário fazer mais trocas. Note que para trocar dois elementos, é necessário usar uma variável auxiliar para que não percamos nenhum dos dois.
Eis o fluxograma:



fluxograma D

Que corresponde a:

```
  10 SLOW
  20 PRINT AT 4,2;"QUANTOS NUMER
OS DEVO ORDENAR ?  (MAXIMO 22)"
  30 INPUT N
  35 CLS
  40 DIM N(N)
  50 FOR I=1 TO N
  60 PRINT AT 4,4;"N(";I;")=?"
  70 INPUT N(I)
  80 NEXT I
  90 FAST
 100 CLS
 105 LET F=0
 110 FOR I=1 TO N-1
 120 IF N(I)<=N(I+1) THEN GOTO 1
70
 130 LET F=1
 140 LET B=N(I)
 150 LET N(I)=N(I+1)
 160 LET N(I+1)=B
 170 NEXT I
 180 IF F=1 THEN GOTO 105
 190 SLOW
 200 FOR I=1 TO N
 210 PRINT TAB 3;N(I)
 220 NEXT I
```

Repare o uso de DIM (linha 20) para reservar espaço na memória para 22 números que deverão ser fornecidos ao computador através de INPUT na linha 70. Note a mudança de velocidades de SLOW para FAST e a seguir para SLOW; isto porque a tela não é necessária enquanto o computador faz a "ordenação". Note também que o loop de ordenação faz o contador I variar de 1 até N-1 pois, caso contrário, ao chegar no último número iríamos obter um erro (Por quê?). Finalmente, repare no uso da variável B como auxiliar para fazer as trocas e o uso do mesmo nome (N) para a variável indexada e para a variável que indica quantos números devem ser ordenados; elas são distinguidas pelo computador pois esta última não possui índice entre parênteses.

Para finalizar o capítulo, apenas uma observação quanto aos índices: eles devem ser sempre números inteiros e positivos.

EXERCÍCIOS

1. Altere o último programa para que imprima uma mensagem de erro se na linha 40 for introduzido um número maior que 22. Além disso, faça com que ele coloque os números em ordem decrescente.

2. Faça um programa que seja capaz de "ler" duas matrizes de duas dimensões e calcule a soma e diferença "imprimindo" na tela a matriz resultado.

3. Elabore um programa que, dada uma matriz tridimensional quadrada (nº de linhas = nº de colunas) calcule seu determinante. Tente usar este programa para fazer outro que seja capaz de resolver um sistema de equações lineares. (Use matrizes de 3 linhas e 3 colunas).

NOTA: dada a matriz:

$$\begin{pmatrix} A & B & C \\ D & E & F \\ G & H & I \end{pmatrix}$$

Seu determinante é dado por:

$$\triangle = A(E*I-H*F)-B(D*I-G*F)+C(D*H-E*G)$$

Dado o sistema linear:

$$\begin{aligned} Ax+By+Cz&=K \\ Dx+Ey+Fz&=L \\ Gx+Hy+Iz&=M \end{aligned}$$

e sendo $\triangle$ o determinante calculado acima, temos:

$$x = \frac{\det\begin{pmatrix} K & B & C \\ L & E & F \\ M & H & I \end{pmatrix}}{\triangle}$$

$$y = \frac{\det\begin{pmatrix} A & K & C \\ D & L & F \\ G & M & I \end{pmatrix}}{\triangle}$$

$$z = \frac{\det \begin{bmatrix} A & B & K \\ D & E & L \\ G & H & M \end{bmatrix}}{\triangle}$$

(det. = determinante)

4. Escreva um programa que aceita uma lista de números inteiros e os imprima em duas colunas separando os pares dos ímpares. Ele deve imprimir uma mensagem de erro se for introduzido um número não inteiro.

5. Faça um programa que seja capaz de construir um triângulo de Pascal dentro de uma matriz 8x8; obviamente o restante de matriz ficará preenchido com zeros.

Neste triângulo, a primeira coluna e a diagonal são constituídos por um número 1.

Os demais elementos são calculados somando-se os vizinhos imediatamente superior e o superior à esquerda (ou vizinhos Norte e Noroeste); veja a figura 7.4.

Nota: Para exemplificar, façamos o triângulo de Pascal até a quarta linha:

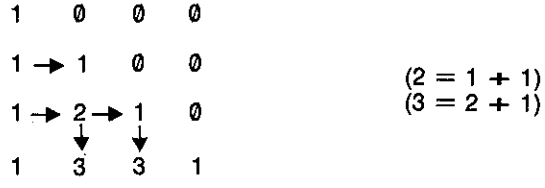
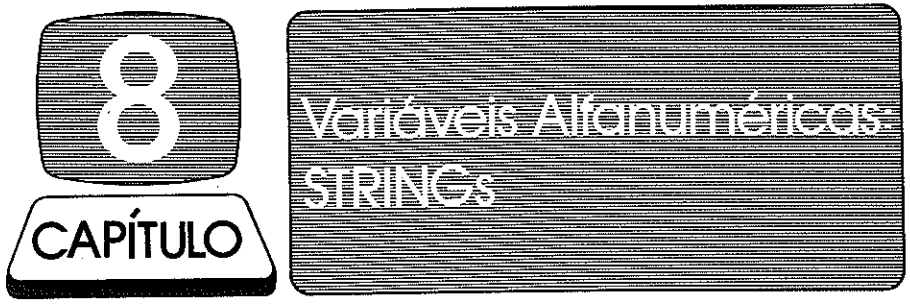


fig. 7.4

6. Faça um programa que leia uma lista de números e calcule sua média aritmética simples. A seguir, complemente o programa para que ele leia os números e seus pesos e calcule uma média ponderada.



Iremos estudar agora como colocar variáveis alfanuméricas na memória. Podemos associar a uma variável caracteres alfanuméricos desde que esta variável tenha seu nome imediatamente seguido pelo símbolo $ (tecla V). Assim, podemos ter:

```
LET L$="FELICIDADE"
```

A este tipo de variáveis, denominamos STRINGs. Note que, além do $, existe a obrigatoriedade de se colocar entre aspas o que desejamos atribuir à variável. Há uma diferença fundamental entre as variáveis numéricas e as STRINGs. No caso das variáveis numéricas:

```
LET L=10

LET L=19956354
```

faz com que apenas *um* espaço de memória seja reservado para colocar lá a variável. Entretanto, para as STRINGs, um espaço de memória é reservado para cada caractere da variável. Assim a L$ anterior ocupa 10 espaços da memória... Em outras palavras, ao definir uma STRING usando LET ou INPUT, o TK faz um DIM "automático", reservando tantos espaços quantos forem o número de caracteres. Quando falamos em caracteres, incluímos os caracteres numéricos, por exemplo fazer:

```
L$="234"
```

reserva três espaços de memória para armazenar os caracteres correspondentes aos símbolos 2, 3 e 4. Em contrapartida, fazer:

```
L=234
```

reserva um espaço de memória para armazenar o valor do número 234. NOTA IMPORTANTE: É bom frisar que estes "espaços de memória" não correspondem a um byte cada, em todos os casos! A ligação entre espaços e número de bytes é razoavelmente complexa e foge dos objetivos deste livro. Naturalmente, o DIM pode também ser utilizado para STRINGs, só que isto cancela o DIM automático feito pelo LET ou INPUT, limitando o tamanho máximo da STRING.

De fato, execute:

```
10 SLOW
20 DIM K$(5)
30 INPUT K$
40 PRINT K$
50 GOTO 30
```

Primeiramente, repare que o cursor L aparece entre aspas, indicando que TKestá aguardando uma STRING.Experimente então entrar com palavras com menos e com mais de 5 letras, para ver o que ocorre. Note que mesmo se a palavra tiver menos de 5 letras, ela continuará ocupando 5 lugares na memória, pois o DIM reservou espaços e os preencheu com caracteres em branco. Repare que se você desejar parar a execução do programa antes de terminar a tela, deverá primeiro eliminar as aspas, o que pode ser feito pressionando SHIFT e EDIT simultaneamente: a seguir, basta pressionar SHIFT e STOP simultaneamente e finalmente NEW LINE. Você pode acessar individualmente cada caracter de sua variável; de fato, após "parar" o programa, experimente escrever a segunda letra da última palavra que você introduziu:

```
PRINT K$(2)
```

Com relação as matrizes de mais de uma dimensão, note a diferença; fazer:

```
DIM P(4,7)
```

reserva espaço na memória para 4 x 7 = 28 números; enquanto que:

```
DIM P$(4,7)
```

reserva 4 x 7=28 espaços correspondentes a 4 palavras com no máximo 7 letras cada uma. Para esclarecer bem as idéias, experimente o programa de introdução de palavras na memória:


```
 10 SLOW
 20 DIM A$(4,7)
 30 FOR I=1 TO 4
 40 PRINT "A$(";I;")=?";
 50 INPUT A$(I)
 60 PRINT A$(I)
 70 NEXT I
 80 PRINT
 90 PRINT A$(2)
100 PRINT A$(2,3)
110 PRINT A$(4,2)
120 PRINT A$(4)
130 STOP
```

Perceba que, ao tratarmos as STRINGs apenas pela primeira coordenada, estamos nos referindo à palavra inteira (linhas 50, 60, 90, 120); assim a linha 50 dentro do loop espera a entrada de 4 palavras de no máximo 7 letras cada e a linha 60 escreve as mesmas na tela. Repare agora nas linhas 100 e 110: a primeira, escreve a terceira letra da segunda palavra e a outra escreve a segunda letra da quarta palavra. Experimente verificar se o TK consegue interpretar matrizes STRING com mais de duas dimensões. Verifique também se é possível que o "nome" de variáveis STRINGs tenham mais que uma letra.

Operações com STRINGs: Slice e adição

Além de podermos acessar individualmente letras de cada palavra, podemos acessar grupos de letras com auxílio de TO (tecla 4), o mesmo utilizado nas instruções de FOR/NEXT. Por exemplo:

```
210 SLOW
220 LET X$="PINDAMONHANGABA"
230 PRINT X$
240 PRINT X$(2 TO 10)
250 PRINT X$(1 TO 5)
260 PRINT X$( TO 5)
270 PRINT X$(8 TO 15)
280 PRINT X$(8 TO )
```

A linha 240 escreve da segunda até a décima letra da variável X$ a 250 da primeira até a quinta e a 270 da oitava linha até a décima quinta. Note que a ausência do primeiro número é interpretado como 1 pelo computador e a ausência do último é interpretado como sendo a posição do último caractere da STRING. Isto é possível também para matrizes bidimensionais. De fato, execute novamente o programa de introdução de palavras na memória acrescentando as seguintes linhas:

```
130 PRINT A$(1,3 TO 5)
140 PRINT A$(3, TO 4)
```

que significam, respectivamente: escreva da terceira até a quinta letra da primeira palavra e escreva da primeira à quarta letra da terceira palavra. A estes tipos de operação, que permitem obter, individualmente, um caractere ou um grupo de caracteres de uma STRING, chamamos *slice*, ou "fatiamento".

Existe ainda outro tipo de operação com STRINGs: a adição. Vamos apresentar um programa que deve ser suficiente para explicar o funcionamento desta operação:

```
10 SLOW
20 LET A$="MORTE"
30 LET B$="VIDA"
40 PRINT A$+B$
50 PRINT B$+A$
60 PRINT B$(4)+A$( TO 3)
70 PRINT B$(4)+A$(3 TO )
80 PRINT "LIN"+B$(3 TO )
90 PRINT "B$"+" "+A$
```

Apenas para dar uma pequena demonstração da utilidade das STRINGs, volte ao capítulo 6, execute o programa que fornece histogramas substituindo as linhas de 40 a 150 por:

```
40 LET X$="JFMAMJJASOND"
50 FOR I=4 TO 26 STEP 2
60 PRINT AT 21,I;X$(I/2)
70 NEXT I
```

As funções CODE, CHR$ e LEN

No TK cada caractere tem um código que corresponde a um número de 0 255. Isto é feito para facilitar o funcionamento do computador, mas isto não poderá ser explicado em detalhes neste livro. Entretanto, há ocasiões em que estes códigos podem ser úteis mesmo em BASIC. Comecemos então pela função CHR$ (tecla V, ela fornece o caractere correspondente ao código especificado. Experimente executar:

```
10 SLOW
20 FOR I=0 TO 260
30 SCROLL
40 PRINT "CHR$ (";I;")=";CHR$
I
50 NEXT I
```


Deste modo, aparecem na tela os caracteres correspondentes aos códigos 0 a 255. Note que o programa para quando a variável, supera 255 (qual a mensagem que aparece no canto inferior esquerdo da tela neste momento?). Repare também que para alguns números não existe código correspondente e o TK coloca então, na tela, um ponto de interrogação. O inverso de CHR$ é a função CODE (tecla I). Ela fornece como saída um número que corresponde ao código do caractere. Se seu argumento for uma palavra, ela fornece o código da primeira letra da palavra. Experimente:

```
10 SLOW
20 LET A$="A"
30 LET B$="2"
40 PRINT A$,CODE A$
50 PRINT B$,CODE B$
60 LET C$=A$+B$
70 PRINT C$,CODE C$
80 LET D$="VALERIA"
90 PRINT D$,CODE D$,CODE D$(4)
```

Note que o valor fornecido por CHR$ é uma STRING e o valor fornecido por CODE é uma variável numérica. Finalmente temos a função LEN (tecla K) que fornece um número correspondente ao número total de caracteres da STRING, ou melhor, o seu "comprimento" de fato:

```
10 SLOW
20 INPUT A$
30 PRINT A$,LEN A$
40 GOTO 20
```

Introduza várias palavras e note que o TK coloca na tela as palavras e o número de caracteres (ou o "comprimento") de cada uma. Vamos então apresentar uma pequena aplicação "prática" que utiliza estas funções. Você pode reparar, com o programa que mostrava todos os caracteres e seu código, que, a diferença entre os códigos de um caractere em vídeo reverso e seu correspondente normal é sempre 128. Assim, podemos fazer o seguinte:

```
10 SLOW
20 INPUT X$
30 LET Y$=""
40 FOR I=1 TO LEN X$
50 LET Y$=Y$+CHR$ (CODE X$(I)+
128)
60 NEXT I
70 PRINT X$,Y$
80 GOTO 20
```

Execute o programa e introduza uma palavra qualquer usando caracteres "normais". Na linha 40 estamos ordenando um *loop* que será repetido um

número de vezes correspondente ao número de caracteres da palavra que você introduziu; a seguir na linha 50, colocamos em Y$ os caracteres de X$ em video-reverso, o que é feito calculando seu código (CODE), somando 128 e achando o caractere correspondente ao código resultante da soma.

As funções VAL e STR$.

Analogamente às funções CODE e CHR$, VAL e STR$ são funções inversas, sendo que a primeira fornece uma variável numérica e a segunda uma STRING. (Repare que a notação utilizada coloca um $ no final do nome das funções que fornecem uma STRING). Estas funções são bastante úteis e permitem interligar os "universos" das variáveis numéricas e das STRINGs pois a primeira (VAL), quando possível, transforma uma STRING no seu valor numérico, e a segunda (STR$) transforma um valor ou uma expressão numérica numa STRING. Experimente o seguinte programa que esperamos ser suficiente para esclarecer o efeito da VAL (tecla J):

```
 10 SLOW
 20 LET B$="2+5-7"
 30 PRINT "B$=";B$
 40 PRINT B$(2 TO 4)
 50 PRINT VAL B$
 60 LET L$="23"
 65 LET Y=VAL L$
 70 LET M$="4"
 80 LET N$="2-47"
 90 PRINT "L$=";L$
 95 PRINT "Y=";Y
100 PRINT "M$=";M$
110 PRINT "N$=";N$
120 PRINT L$+M$
130 PRINT Y+VAL M$
140 PRINT M$+N$
150 PRINT VAL (M$+N$)
160 LET X=6
170 LET A$="X+3"
180 PRINT "X=";X
190 PRINT "A$=";A$
200 PRINT VAL A$
210 LET A$="5*RND"
220 PRINT VAL A$
```

Estude cuidadosamente todas as saídas do programa. Repare novamente a diferença de "espaços" de memória ocupados; por exemplo, na linha 60 temos L$="23" ocupando dois espaços, um para cada caractere,e na linha 65 temos Y =VAL L$,ou seja, 23 também, mas agora ocupando apenas um espaço de memória. Em outras palavras, L$ é uma matriz de duas posições enquanto que Y é uma variável simples, apesar de ambas produzirem o mesmo efeito na tela... Preste atenção agora na linha 210; se você digitou a tecla T com o cursor em F tudo bem e a linha 220 será executada; no entanto se você digitou as letras R, N

e D separadamente o programa acusará um erro na linha 220. Por quê? Outra utilidade do VAL está no cálculo de expressões ou funções matemáticas. Por exemplo:

```
 10 SLOW
 20 PRINT "ENTRE COM UMA FUNCAO
MATEMATICA QUE USE A VARIAVEL X
"
 30 INPUT Z$
 40 PRINT Z$
 50 PRINT
 60 PRINT "ENTRE COM O VALOR DA
VARIAVEL X"
 70 INPUT X
 75 PRINT X
 80 PRINT
 90 PRINT "VALOR CALCULADO"
100 PRINT VAL Z$
```

Experimente entrar com 5*X**2-2*X+7" e 5 para X; o que você deverá obter? Finalmente, apresentamos um exemplo mostrando uma utilização de STR$ (tecla Y) para obter o número de algarismos de um determinado número inteiro e obter alguns de seus dígitos separadamente:

```
 10 SLOW
 20 LET X=12345678
 30 PRINT "X=";X
 40 LET A$=STR$ X
 50 PRINT "A$=";A$
 60 PRINT LEN A$
 70 PRINT A$(4)
 75 PRINT X
 80 PRINT A$(2 TO 6)
 90 PRINT A$(1)+A$(3)+A$(5)
```

A variável X ocupa um espaço de memória para armazenar o número 12.345.678 na linha 40 criamos a variável A$ (que ocupa 8 espaços de memória) para armazenar os caracteres 1,2,3,4,5,6,7 e 8.

Procure imaginar como seria complicado calcular o número de algarismos de cada variável aritmética ou "extrair" individualmente seus algarismos se não tivéssemos essas funções.

Um jogo simples utilizando STRINGs: Forca

Vamos apresentar agora, uma versão bastante simplificada do jogo de forca, que apresenta algumas limitações mas é bastante útil no sentido de utilização de variáveis STRINGs.

```
 10 SLOW
 20 PRINT TAB 2;"QUAL A PALAVRA
?"
 30 INPUT P$
 40 CLS
 50 LET T=LEN P$
 60 FOR I=1 TO T
 70 PRINT AT 5,2*I;"▒"
 80 NEXT I
 90 LET C=0
100 PRINT AT 10,2;"LETRA=?"
110 INPUT L$
120 PRINT AT 10,2;"          "
130 FOR I=1 TO T
140 IF P$(I)<>L$ THEN GOTO 170
150 LET C=C+1
160 PRINT AT 5,2*I;L$
170 NEXT I
180 IF C<>T THEN GOTO 100
190 PRINT AT 10,2;"FIM DE JOGO"
```

Você deve introduzir uma palavra (linha 30) e fazer com que outra pessoa tente "adivinhar" a mesma. Note entretanto que nessa versão do jogo você nunca perde... Estude com cuidado este programa e repare no uso da variável C como contadora, para verificar se você já acertou todas as letras; repare também na formação das linhas 70 e 160. Outro fato interessante é a possibilidade de se comparar duas STRINGs: veja a linha 140... Neste ponto você poderia se perguntar se, ao comparar duas STRINGs, pode-se falar em maior ou menor, além de igual ou diferente. Felizmente os códigos dos caracteres no TK estão colocados em ordem crescente para as letras, seguindo a ordem do alfabeto; ao comparar STRINGs o computador compara seus códigos. Assim a letra A é "menor" que a letra B. Além disso, a palavra "BALA" é "menor" que "BOLA".

De posse dessas informações, experimente criar um programa a que coloque em ordem alfabética uma lista de palavras fornecidas por você. Perceba como estas funções que lidam com STRINGs podem facilitar enormemente o uso de arquivos de dados.

OBS: Experimente comparar a STRING "A" com a STRING "9" (caractere nove) para ver quem é "maior"...

NOTA: No capítulo 5 mostramos uma limitação da impressora do TK; usando STRINGs, ela pode ser eliminada. De fato, experimente:

```
10 LET X=0.00001
15 LET X$=STR$ X
20 LPRINT X$
30 LPRINT X
```


A função INKEY$

O TK possui uma função bastante interessante chamada INKEY$ e que é útil para a realização de jogos animados; seu efeito é similar ao INPUT, ou seja, ela aceita entrada de dados através do teclado; entretanto, ela só aceita STRINGs formados por apenas um caractere. Em outras palavras, ela fornece a STRING correspondente à tecla que está sendo pressionada, além disso, ao contrário de INPUT, ela não interrompe a execução do programa. Se, no instante em que a instrução que contém INKEY$ está sendo executada, uma tecla é pressionada, ela "entra" no computador; caso contrário, o programa prossegue. Experimente então o seguinte programa (INKEY$ está na tecla B):

```
10 SLOW
20 PRINT "*";
30 PRINT INKEY$;
40 GOTO 20
```

Digite RUN e NEW LINE. Começarão a ser impressos asteriscos. Antes de acabar a tela, digite qualquer tecla (pode ser, inclusive, NEW LINE) e tente explicar o que ocorre.

Vamos agora "complicar" um pouco mais o exemplo:

```
10 SLOW
20 LET X=31
30 LET Y=21
40 PLOT X,Y
50 IF INKEY$="5" THEN LET X=X-
1
60 IF INKEY$="8" THEN LET X=X+
1
70 IF INKEY$="6" THEN LET Y=Y-
1
80 IF INKEY$="7" THEN LET Y=Y+
1
90 GOTO 40
```

Ao executar o programa aparecerá um ponto no centro da tela. Experimente movimentá-lo utilizando as teclas 5, 6, 7 e 8, que correspondem aos quatro sentidos indicados pelas setas, que aparecem nestas teclas. Em capítulos futuros, veremos maiores detalhes sobre a função INKEY$. O que acontece se você tentar "sair" da tela?

A utilização das STRINGS na aritmética financeira

Como você deve ter notado, o BASIC possui o inconveniente de colocar os números encostados à esquerda e com várias casas depois do ponto decimal. Ao se trabalhar em aritmética financeira é norma não se colocar os milhões na

mesma coluna dos milhares, por exemplo. Além disso, não tem sentido apresentar os dados com precisão de até milésimos de cruzeiros (ou menos!). Também é norma apresentar-se os valores com um ponto separando os algarismos de três em três e com uma vírgula separando os cruzeiros dos centavos. Tudo isso é possível de ser feito utilizando STRINGs. De fato:

```
 10 FAST
 20 PRINT "ENTRE COM O NUMERO N
O FORMATO   CONVENCIONAL DO TK"
 30 INPUT N
 40 PRINT N
 50 PRINT
 60 LET N$=STR$ N
 70 LET L=LEN N$
 80 FOR U=1 TO L
 90 IF N$(U)="." THEN GOTO 120
100 NEXT U
110 LET N$=N$+",00"
120 LET N$(U)=","
130 LET N$=N$+"0"
140 LET L=U+2
150 LET N$=N$( TO L)
160 LET L=L-3
170 LET L=L-3
180 IF L<=0 THEN GOTO 230
190 LET C$=N$( TO L)
200 LET N$(L)="."
210 LET N$=C$+N$
220 GOTO 170
230 PRINT "CR$ ";TAB (21-L);N$
235 PRINT
240 GOTO 20
```

Experimente executar o programa para diferentes entradas: 1954.3 ; 20.614.412; 347, etc...

Note que os números aparecem no formato padrão e encostados à direita na 21ª coluna. Em capítulos futuros voltaremos novamente a este programa para uma utilização mais geral. Por enquanto, tente estudar, com detalhes, todos os "truques" aqui utilizados.

EXERCÍCIOS

1) Modifique o programa que lê os caracteres de uma STRING e os escreva em vídeo reverso, para que ele aceite STRINGs que tenham caracteres normais e em vídeo-reverso, simultaneamente.

2) Elabore um programa que leia uma STRING e a escreva na tela, de trás para frente, em vídeo normal e reverso.

Exemplo:

ROMA

AMOR

3) Faça um programa que seja capaz de ler uma STRING e a seguir apresentar na tela a STRING "rodando", como nos letreiros luminosos para anúncios ou jornais:

```
BEATLES
SBEATLE
ESBEATL
LESBEAT
TLESBEA
ATLESBE
EATLESB
```

4) Neste capítulo foi sugerido que você fizesse um programa que ordenasse, em ordem alfabética, uma lista de palavras. Implemente então um programa para que, uma vez feita a lista, ele possibilite a inserção ou retirada de palavras da mesma.

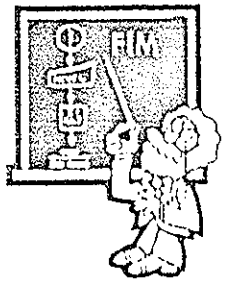
5) Agora, com relação ao jogo da forca, implemente-o para que ele limite o "número de tentativas" para que seja possível perder o jogo. Tente também fazer com que ele não aceite letras que já foram digitadas e que não aceite também mais de uma letra por vez. Quem sabe você consegue chegar até a desenhar a forca...

6) Modifique o programa que movimenta um ponto na tela para que, usando as teclas 3, 4, 9 e 0 você consiga movimentar o ponto nas diagonais. A seguir, faça com que a saída apresente linhas tracejadas em vez de contínuas.

7) Modifique o programa que formata números para contabilidade para que ele trabalhe também com números negativos.

8) Faça um programa que ajudará você a treinar sua velocidade de fazer contas: primeiramente, ele deve perguntar que operação você deseja fazer (+,–,*ou/), o número de operações, o número máximo para utilizar nas contas e o *tempo* que você deseja para responder (naturalmente este tempo não poderá ser contado nas unidades convencionais). A seguir, o TK deve gerar uma série de operações com números aleatórios inteiros, compreendidos entre 1 e o número máximo que você estipulou; para cada operação, o "tempo" pode ser contado através de um *loop* que testa continuamente o INKEY$: se o INKEY$ for, por exemplo, "X", significa que você já sabe a resposta e deve fornecê-la ao computador; se terminar o *loop* e você não apertou X, ele imprime tempo esgotado e passa para a próxima operação. Para cada operação respondida ele

deve colocar "ACERTOU" ou "ERRROU" seguida da resposta correta e, no final do programa, sua nota de 0 a 10 seguida de comentários sobre sua "performance".
OBS: Raciocine em "diagrama de blocos" para os programas mais complexos.



Funções Matemáticas

Neste capítulo, iremos estudar a "potencialidade" matemática do TK. Comecemos recordando a função ABS (tecla G) que fornece o "módulo" de seu argumento, assim:

```
ABS (-3)=3

ABS (4.5)=4.5
```

Eis um pequeno exemplo de aplicação:

```
10 SLOW
20 FOR I=20 TO -22 STEP -2
30 PRINT TAB ABS I;"*"
40 NEXT I
```

As funções logarítmicas e exponenciais

Vamos agora às funções que lidam com logaritmos e exponenciais; temos LN (tecla Z) que calcula o logaritmo natural (na base e ≃ 2.72) e EXP (tecla X) que calcula a potência na base natural (e) de seu argumento, ou seja, e "elevado" ao número fornecido. Convém lembrar que LN e EXP são funções inversas; de fato, experimente ordenar:

```
PRINT LN EXP 1
```

O que você espera obter na tela?

Vejamos então as funções trigonométricas; temos as 3 funções básicas e seus respectivos inversos:

Seno	SIN (tecla Q)	Arco seno	ARC SIN (tecla A)
Cosseno	COS (tecla W)	Arco cosseno	ARC COS (tecla S)
Tangente	TAN (tecla E)	Arco tangente	ARC TAN (tecla D)

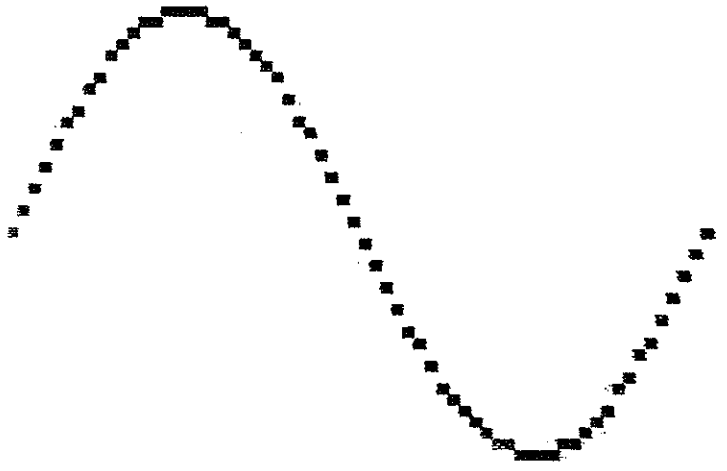
Atenção: as funções SIN, COS e TAN presumem que o ângulo esteja em *radianos*! Assim, para facilitar, o TK já tem o número PI (π) em sua memória (tecla M); note que apesar de na tecla correspondente estar desenhado o símbolo π, ao pressioná-la ele aparecerá na tela como PI (Vale a pena lembrar que *não adianta* escrever as letras P - I individualmente).

Obs: π radianos correspondem a 180°.

O que acontece se você tentar calcular o seno (ou cosseno ou tangente) de um ângulo maior que 2π?

Vamos então, a título de exemplo, fazer o gráfico da função seno. Como esta função possue também valores negativos e como a "origem" das coordenadas no TK situa-se no canto esquerdo da tela, deveremos fazer um "deslocamento de eixos", assim:

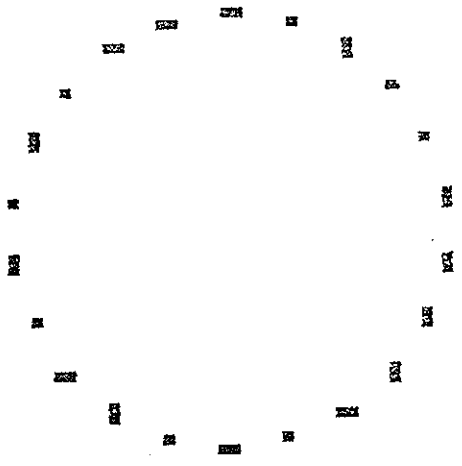
```
10 SLOW
20 PRINT "FREQUENCIA=?"
30 INPUT W
40 CLS
50 FOR I=0 TO 63
60 LET J=20*SIN (2*W*(I/63)*PI

70 PLOT I,J+22
80 NEXT I
```

Experimente usar valores de freqüência cada vez maiores: 0.5, 1, 2, 3, 4, etc., até chegar no máximo de inteligibilidade do gráfico. Note o truque utilizado na linha 60 para fazer com que, para W=1,coubesse exatamente um período da senóide na tela. Por que é necessário a multiplicação por 20?

Utilizaremos agora as funções trigonométricas para desenhar um "círculo" na tela:

```
 10 FAST
 20 PRINT "FREQUENCIA=?"
 30 INPUT W
 40 PRINT W
 50 PRINT "PASSO=?"
 60 INPUT P
 70 PRINT P
 80 PRINT "FINAL=?"
 90 INPUT F
100 PRINT F
109 PAUSE 120
110 CLS
120 FOR I=0 TO F STEP P
130 LET X=20*SIN (W*I)
140 LET Y=20*COS (W*I)
150 PLOT X+31,Y+21
160 NEXT I
```



O que usamos para definir a circunferência foi um conjunto de equações que calcula a posição de um ponto qualquer que pertença a ela. Estas equações são chamadas de paramétricas.

Para iniciar, utilize valores tais que W>1, P<.8 e F>50. Estude com cuidado este programa, notando o deslocamento de eixos na linha 150. Qual o raio desta circunferência? (*Obs*: o programa, apesar de ser em FAST, é um pouco demorado pois não é tão simples calcular vários senos e cossenos...)

Espirais

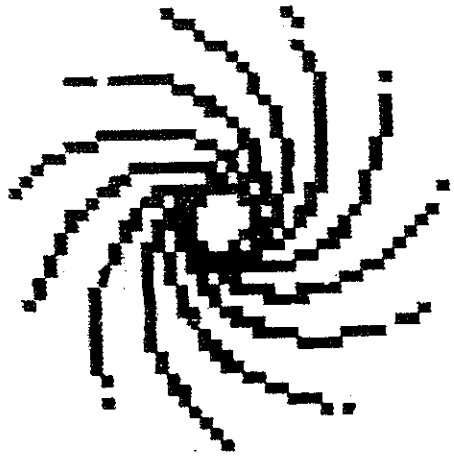
Acrescentando algumas linhas ao programa anterior, poderemos ter uma série de efeitos realmente bonitos. Coloque então as seguintes instruções adicionais:

```
102 PRINT "DECAIMENTO=?"
104 INPUT D
106 PRINT D
125 LET A=EXP (-D*I)
```

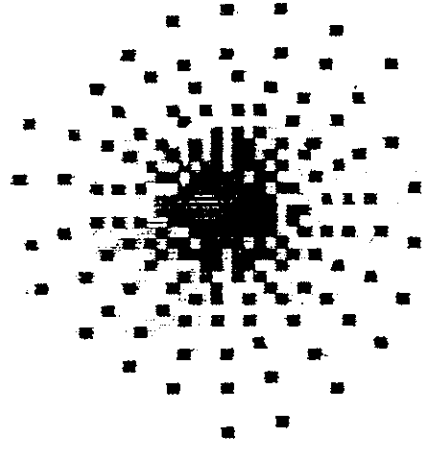
e modifique as linhas 130 e 140 para:

```
130 LET X=20*A*SIN (W*I)
140 LET Y=20*A*COS (W*I)
```


Experimente então, para começar, W = 120, P = 0.1, F = 50 e D = 0.04.

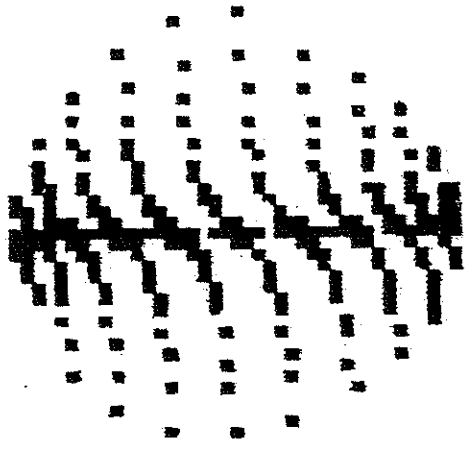


A seguir tente W = 10, P = 0.2, F = 100 e D = 0.05.



Finalmente, para estes mesmos valores, experimente retirar A da linha 130.

```
130 LET X=20*SIN (W*I)
```



Procure explicar o funcionamento desse programa. Tente, inclusive, acrescentar uma "defasagem" nos argumentos dos SIN e COS para tornar as figuras formadas mais estranhas...

EXERCÍCIOS

1. Implemente o programa do gráfico do seno para que ele também desenhe o eixo das abscissas (eixo X).

2. Fazendo com que o eixo X esteja na linha "mais baixa" da tela, faça o gráfico das seguintes funções:

a) f(x)=|40 senx|

b) f(x) = e^{-Dx} *|40 Senx|

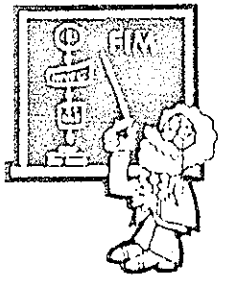
Obs: |x|=módulo de x; em b, faça o gráfico para vários valores de *D*.

3. Faça um programa para converter graus em radianos; a seguir, implemente-o para que ele consiga calcular o seno, cosseno, tangente, cotangente, secante e cossecante, de um ângulo fornecido em graus.

4. Elabore um programa que calcula o logaritmo de uma variável na base 10.

5. Faça um programa que seja capaz de "ler" uma função matemática (usando STR$) e calcule, dentro de um dado intervalo com erro máximo definido, as raízes da função.

6. Elabore um programa que, de posse da altura de um farol marítimo e do ângulo que a luz deixa o mesmo para atingir um navio, calcule a distância entre o navio e o farol.

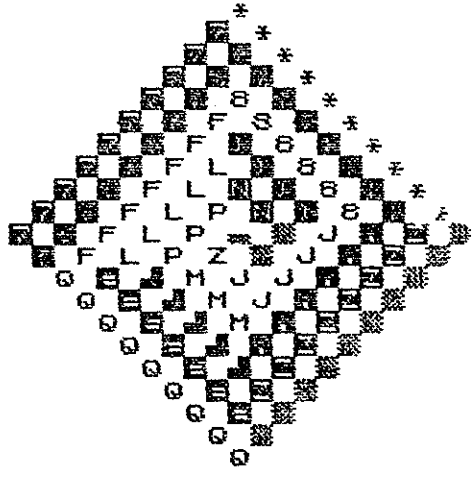


10 CAPÍTULO

As sub-rotinas e a gravação de programas em fita

Quando uma sequência de instruções deve ser repetida diversas vezes num programa, ela pode ser escrita apenas uma vez como se fosse um pequeno programa separado. Este programa deve ser "chamado" pela palavra-chave GOSUB (tecla H) e deve obrigatoriamente terminar com uma palavra-chave que faça com que o programa volte à linha logo após o GOSUB. Esta palavra chave é RETURN (tecla Y). A esta sequência de instruções chamamos *sub-rotina*. Repare no seguinte programa:

```
 10 SLOW
 20 LET C=1
 30 GOSUB 280
 40 FOR P=1 TO 11
 50 PRINT AT 12-P,P+5;CHR$ X
 60 NEXT P
 70 FOR N=1 TO 11 STEP 2
 80 GOSUB 280
 90 FOR P=1 TO 12-N
100 PRINT AT -1+P+N,15+P;CHR$ X
110 NEXT P
120 GOSUB 280
130 FOR P=1 TO 12-N
140 PRINT AT 10+P,25-P-N;CHR$ X
150 NEXT P
160 GOSUB 280
170 FOR P=1 TO 11-N
180 PRINT AT 23-P-N,17-P;CHR$ X
190 NEXT P
200 GOSUB 280
210 FOR P=1 TO 11-N
220 PRINT AT 13-P,5+P+N;CHR$ X
230 NEXT P
240 NEXT N
250 LET C=C*(-1)
260 LET X=0
270 GOTO 30
280 IF C=1 THEN LET X=1+INT (12
7*RND)
290 IF X>63 THEN LET X=X+64
300 RETURN
```



Note que nas linhas 30, 80, 160 e 200 "chamamos" a mesma seqüência de instruções (sub-rotinas) que estão nas linhas 280 e 290, terminando com um RETURN. Assim ao chegar na linha 30, o TK passa para as linhas 280, 290 e 300 e volta para a linha 40; ao chegar na linha 80, ele vai novamente para as linhas 280, 290 e 300, mas volta agora para a linha 90. E assim por diante.

As sub-rotinas têm um símbolo especial para uso em diagramas de bloco; veremos isto logo mais neste mesmo capítulo.

Animação universal usando PRINT e PLOT - a palavra-chave - REM

À medida que os programas se tornam mais complicados, seria útil que pudéssemos colocar alguns "comentários" junto à listagem para esclarecer melhor os detalhes. Além disso, seria bom se pudéssemos dar *nomes* aos programas. Para isto, temos a palavra-chave REM (tecla E) que não executa nada; apenas permite a inserção de comentários nos programas.

Observe o seguinte exemplo:

```
   1 REM PIAZZI-ROSSINI
  10 REM PROGRAMA ANIMACAO UNIVE
RSAL - PRINT AT
  20 SLOW
  30 FOR H=5 TO 14
  40 PRINT AT H,10;"▓    ▓"
  50 NEXT H
  60 PRINT AT 15,10;"▓▓▓▓▓▓"
  70 REM LINHAS EM L$
  80 LET L$="0011223344556677889
9AABBCCDDEEDDCCBBAA998877665544З
3221100"
  90 REM COLUNAS EM C$
 100 LET C$="778899AABBCCDDEEDDC
CBBCCDDEEDDCCBBCCDDEEDDCCBBCCDDE
EFFGGHH"
 110 REM CARACTERES EM X$
 120 LET X$="▒ ▙ ▜ ▀ ▚ █ ▙ ▟ ▓ ▓
 ▓ ▓ ▓ ▓ ▓ ▓ ▟ ▓ ▜ ▓ ▚ █ ▙ ▓ ▓ ▓
 ▓ ▓ ▓"
 130 REM TEMPOS EM T$
 140 LET T$="6666666666111111111
11111111111111111111111111111111
1222222"
 150 IF LEN L$<>LEN C$ OR LEN C$
<>LEN X$ OR LEN X$<>LEN T$ THEN
STOP
 160 FOR P=1 TO LEN X$
 170 PRINT AT CODE L$(P)-28,CODE
 C$(P)-28;X$(P)
 180 FOR T=1 TO CODE T$(P)-28
 190 NEXT T
 200 NEXT P
```

Utilizamos então as linhas REM para esclarecer os vários pontos ao programa. De fato, representamos um objeto que se move mudando continuamente de forma, colocando as linhas e colunas para PRINT, AT nas STRINGs L$ e C$ respectivamente, os vários caracteres em X$ e os "tempos" de duração de cada movimento em T$.

Repare na utilização do *código* dos caracteres de tal maneira que uma correspondência pode então ser feita segundo a tabela 10

Tabela 10

código	linha	coluna	tempo	
0	0			
1	1			
2	2			
3	3			
4	4			
5	5			
6	6			
7	7			
8	8	L		
9	9	I		
A	10	N		
B	11	H		
C	12	A	C	
D	13	S	O	
E	14		L	
F	15		U	T
G	16		N	E
H	17		A	M
I	18		S	P
J	19			O
K	20			
L	21			
M	22			
N	23			
O	24			
P	25			
Q	26			
R	27			
S	28			
T	29			
U	30			
V	31			
W	32			
X	33			
Y	34			
Z	35			

Pode-se fazer um programa equivalente também para PLOT:

```
  1 REM PIAZZI-ROSSINI
 10 REM PROGRAMA ANIMACAO UNIVE
RSAL - PLOT
 20 REM COORDENADAS X/Y  +PLOT
-UNPLOT
 30 SLOW
 40 LET C$="09+30 07+32 08+32 1
0+32 12+32 14+32 16+32 18+32 19+
32 09-30 09+31 09-31 09+32 09+33
 09-33 10+33 10-33 11+33 11-33 1
1+32 11+31 11-31 12+31 12-31 13+
31 13-31 14+32 13+33 13-33 14+33
 14-33 15+33 15-33 15+32 15+31 1
5-31 16+31 16-31 17+31 17-31 17+
32 17+33 17-33 17+34"
 50 FOR P=1 TO LEN.C$ STEP 6
 60 IF CODE C$(P+2)=21 THEN PLO
T VAL C$(P TO P+1),VAL C$(P+3 TO
 P+4)
 70 IF CODE C$(P+2)=22 THEN UNP
LOT VAL C$(P TO P+1),VAL C$(P+3
TO P+4)
 80 NEXT P
```

Aqui colocamos as coordenadas para PLOT ou UNPLOT na STRING C$, diferenças por + para PLOT e - para UNPLOT. Você saberia colocar o efeito de "tempo" neste programa analogamente ao programa anterior? Estude detalhadamente estes dois últimos programas antes de prosseguir.

ZAPPLENUM

Faremos agora um pequeno "jogo" que demonstra a utilização de muitos comandos e estruturas do BASIC e inclusive um truque novo com relação ao INKEY$;

Vejamos:

```
  1 REM PIAZZI-ROSSINI
 10 REM ZAPPLENUM
 20 SLOW
 30 RAND
 40 PRINT "QUANTOS NUMEROS ?"
 50 INPUT N
 60 PRINT "TEMPO EM ZUTUS ?"
 70 INPUT T
 80 PRINT "DIFICULDADE (0,1,2)?
"
 90 INPUT D
100 CLS
```



```
104 LET X=10
106 LET Y=X
110 LET S=0
120 PRINT "SCORE ";S;TAB 9;"ZAP
PLENUM"
130 FOR I=1 TO N
140 LET A=INT (21*RND)+1
150 LET B=INT (31*RND)
160 PRINT AT A,B;I
170 FOR J=1 TO T-D*I
180 PRINT AT X,Y;"■"
190 PRINT AT X,Y;" "
200 LET Y=Y+(INKEY$="8")-(INKEY
$="5")
210 LET X=X+(INKEY$="6")-(INKEY
$="7")
220 IF X<1 THEN LET X=21
230 IF X>21 THEN LET X=1
240 IF Y<0 THEN LET Y=31
250 IF Y>31 THEN LET Y=1
260 PRINT AT 0,20;"[illegible]"
270 IF X=A AND Y=B THEN GOSUB 3
50
280 PRINT AT 0,20;"BEATLES"
290 NEXT J
300 NEXT I
310 CLS
320 PRINT AT 10,10;"FIM DE JOGO
"
330 IF S>50 THEN PRINT AT 13,12
;"MUITO BEM..."
340 STOP
350 LET S=S+I
360 PRINT AT 0,6;S
370 FOR K=1 TO 15
380 PRINT AT 0,20;"NHAC..."
390 PRINT AT 0,20;"[illegible]"
400 NEXT K
410 PRINT AT 0,20;"
420 LET J=T-D*I+2
430 RETURN
```

O TK gera números em posições aleatórias de tela que devem ser "comidos" por você(■)antes que apareça algum novo número; assim, por exemplo, se você estiver perseguindo o número quatro, antes de alcançá-lo, aparece o número cinco na tela, não adianta mais "comer" o número quatro pois seu SCORE não será alterado. Quanto maior for o número, menor o tempo que você terá para "comê-lo", em contrapartida mais pontos você ganha se conseguir. Se for gerado um número com dois algarismos, você deve "comer" o mais significativo (Por quê?) Brinque então um pouco com ele, usando, para começar, 10 números, tempo 18 ZUTUS e nível 1 de dificuldade. O tempo é contado numa unidade qualquer (ZUTUS - Zapple Universal Time Unities) pois a relação com as unidades convencionais não é simples. Neste programa é interessante salientar as linhas 200 e 210: quando aparece mais do que uma igualdade (e/ou desigualdade) numa expressão matemática, apenas a primeira,

contando da esquerda para a direita, é considerada como tal. As demais são consideradas como parte de uma afirmação lógica que pode ser verdadeira ou falsa: se verdadeira, produz resultado 1 e, se falsa, produz resultado zero. Assim, na linha 200, se, no instante em que o TK estiver executando a instrução, a tecla 8 estiver pressionada (→), a primeira afirmação é verdadeira e a segunda é falsa; assim teremos:

```
200 LET Y=Y+1-0
```

Note que este tipo de estrutura permite o uso das operações lógicas AND, OR e NOT; assim, por exemplo, seria válido escrever:

```
LET H=H+(INKEY$<>"8" AND I=5)
```

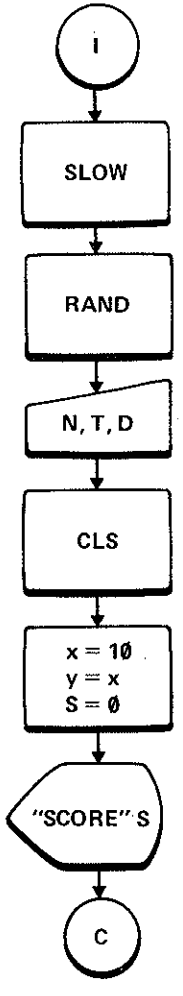
ou

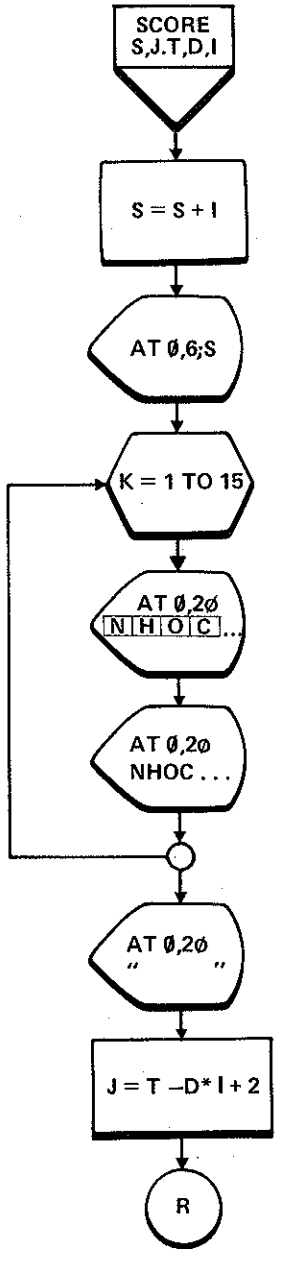
```
LET P=P-5*(A<=3 OR K=A+B)
```

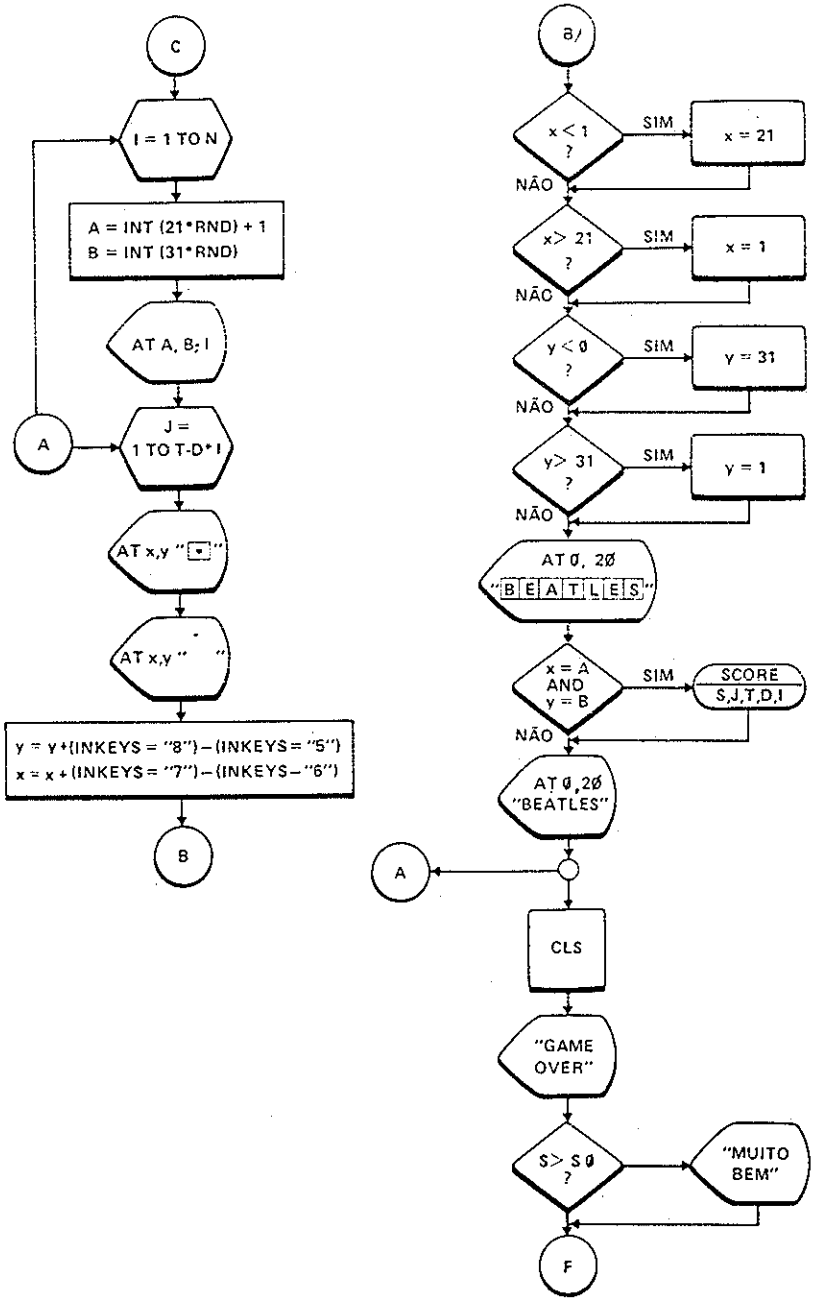
Para entender melhor, procure fazer alguns comandos diretos do tipo:

```
PRINT (3<4 OR 4<3)

PRINT (9=9 AND 7>5)

PRINT (NOT (2<1))

PRINT (3=3)

PRINT (6>10)
```

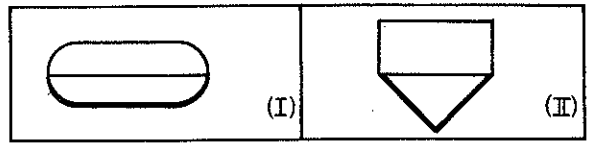
Quanto ao resto do programa, acreditamos que você possa entendê-lo sem dificuldades se dedicar um pouquinho de tempo sobre sua lógica. Note nos detalhes como o "pisca-pisca" das palavras BEATLES e NHOC... Note também que a sub-rotina utilizada não precisava necessariamente ser uma sub-rotina; bastaria fazer GOTO 350 ao invés de GOSUB (na linha 270) e GOTO 280 ao invés de RETURN (na linha 430). Para facilitar seu raciocínio, iremos apresentar o fluxograma do programa:







Note o símbolo utilizado para representar "chamadas" de sub-rotinas (I) e para iniciar as sub-rotinas em si (II). Costuma-se associar um *nome* à sub-rotina o qual é colocado na parte superior do símbolo e, na parte inferior, colocam-se as variáveis do programa principal que serão utilizadas pela sub-rotina. Você saberia explicar para que serve a instrução da linha 420?



Sub-rotinas práticas: aritmética financeira

Como prometido no capítulo 8, voltamos a falar sobre aritmética financeira. Reapresentaremos o programa de formação de dados em forma de uma sub-rotina para "saída de dados com formato financeiro" além de outra sub-rotina para "entrada de dados no formato financeiro". Vejamos:

```
 995 REM SR P/ ENTRADA DE DADOS
       C/ FORMATO FINANCEIRO
1000 FAST
1005 INPUT A$
1010 LET L=LEN A$
1015 DIM B$(L)
1020 FOR U=1 TO L
1025 IF A$(U) <> "." THEN LET B$(U
) =A$(U)
1030 IF A$(U) ="," THEN LET B$(U)
=" "
1035 NEXT U
1040 LET N=VAL B$
1045 SLOW
1050 RETURN
```

Esta sub-rotina simplesmente retira todos os pontos do dado introduzido e substitui a vírgula por um ponto, deixando o valor número para cálculos na variável N.

A sub-rotina para executar a "saída" de dados, ou seja, transformar um número no formato do computador para o formato financeiro, é bem mais complexa. Ela consta de três módulos básicos:

a) ajustar o número para duas "casas" após o ponto decimal (que será substituído por uma vírgula) para os três casos possíveis: nenhuma casa após o ponto decimal (variável inteira), uma casa e duas ou mais casas após o ponto decimal.

b) Colocar um ponto entre cada 3 algarismos antes da vírgula.

c) Imprimir o número encostando-o à direita na vigésima coluna.

```
1095 REM SR P/ SAIDA DE DADOS
       C/ FORMATO FINACEIRO
1100 FAST
1105 REM MODULO A
1110 LET N$=STR$ N
1115 LET L=LEN N$
1120 FOR U=1 TO L
1125 IF N$(U)="." THEN GOTO 1140
1130 NEXT U
1135 LET N$=N$+".00"
1140 LET N$(U)=","
1145 LET N$=N$+"0"
1150 LET L=U+2
1155 LET N$=N$( TO L)

1160 REM MODULO B
1165 LET L=L-3
1170 LET L=L-3
1175 IF L<=0 THEN GOTO 1210
1180 LET C$=N$( TO L)
1185 LET N$(L)="."
1190 LET N$=N$(L TO )
1195 LET N$=C$+N$
1200 GOTO 1170

1205 REM MODULO C
1210 PRINT "CR$ ";TAB (21-L);N$
1220 SLOW
1225 RETURN
```

O número a ser calculado deve ser colocado na variável N. A título de exemplo, vamos apresentar um pequeno programa que utiliza estas sub-rotinas; você fornece o preço de custo de três produtos e a porcentagem de lucro desejada para cada um e ele fornece o preço de venda e o lucro...

```
  10 REM EXEMPLO DE APLICACAO P/
       SUBROTINAS FINANCEIRAS

  20 SLOW
  25 LET ENTFIN=1000
  30 LET SAIFIN=1100
  35 DIM P$(3,14)
  40 DIM K(3,2)
  45 FOR I=1 TO 3
  50 PRINT "NOME DO PRODUTO ";I;
" ?"

  55 INPUT P$(I)
  60 PRINT P$(I)
  65 PRINT "PRECO DE CUSTO ?"
  70 GOSUB ENTFIN
  75 LET K(I,1)=N
  80 PRINT "CR$ ";A$
  85 PRINT "PORCENTAGEM DE LUCRO
 ?"
  90 GOSUB ENTFIN
  95 LET K(I,2)=N
 100 PRINT A$;" POR CENTO"
 105 PAUSE 20
 110 CLS
 115 NEXT I
 120 FOR I=1 TO 3
 125 LET LUCRO=(K(I,2)*0.01)*K(I
 1)
 130 PRINT "PRODUTO ";P$(I)
 135 LET N=K(I,1)+LUCRO
 140 PRINT TAB 5;"PRECO DE VENDA
"

 145 GOSUB SAIFIN
 150 LET N=LUCRO
 155 PRINT TAB 5;"LUCRO"
 160 GOSUB SAIFIN
 165 PRINT
 170 NEXT I
 175 PRINT
 180 PRINT "ALL YOU NEED IS LOVE
"

 185 PRINT
 190 PRINT "BEATLES FOREVER"
```

Experimente desenhar o fluxograma destes programas.

O gravador K-7

Você deve ter notado como não é muito prático ter que digitar um programa "grande", especialmente quando isto já foi feito diversas vezes... Felizmente o TK possibilita a gravação de programas em fita K-7 comum para posterior recuperação. Para colocar um programa no gravador, utiliza-se a palavra chave SAVE (letra S), dando-se um nome ao programa (o qual deve ser colocado entre aspas). Suponha que tenhamos um programa chamado CHOCOLATE. Conecte então a saída MIC do seu TK à entrada MIC do seu gravador, ajuste o volume para aproximadamente 70% do nível máximo e, se tiver controle de tonalidade, coloque o mais agudo possível. A seguir, pressione as teclas do gravador para gravar e digite o seguinte comando direto:

```
SAVE "CHOCOLATE"
```

A tela ficará "esquisita" por alguns instantes; quando ela voltar ao "normal" a gravação está terminada. É recomendável que, por segurança, você

grave um mesmo programa diversas vezes na fita.

Para recuperar o programa da fita K-7 a operação também é simples. Suponha que você tenha realmente gravado um programa chamado CHOCOLATE. Digite NEW para limpar a memória, volte a fita e conecte a entrada EAR do seu TK à saída EAR do seu gravador e com os controles de volume e tonalidade idênticos à operação de SAVE, coloque o seu gravador para "tocar" após digitar o seguinte comando direto:

```
LOAD "CHOCOLATE"
```

(O LOAD está na tecla J). Se após a tela "esquisita" aparecer a mensagem 0/0 você conseguir recuperar o programa; caso contrário, repita a operação!

Aqui vai uma dica; experimente digitar o seguinte:

```
10 PRINT "LOVE IS THE ANSWER "

20 GOTO 10
30 SAVE "LOVE"
40 RUN
```

Prepare o gravador para um SAVE e digite o seguinte comando direto:

```
GOTO 30
```

O Programa será armazenado na fita... Mas, ao carregá-lo de volta, ele começa a sua execução sem ser necessário digitar RUN e NEW LINE; experimente, então, voltar a fita e fazer:

```
LOAD "LOVE"
```

EXERCÍCIOS

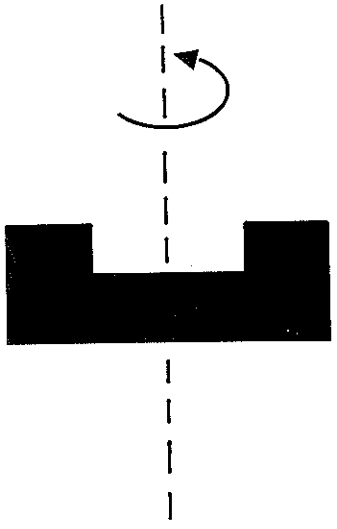
1. Implemente os três últimos programas para que eles possam trabalhar também com números negativos. Além disso, note que eles "truncam" o resultado em duas casas após a vírgula; faça com que eles arredondem o resultado dependendo do eventual valor da terceira casa após a vírgula. Sugestão: Após a entrada no módulo B do segundo programa, teste VAL N$ (L+1) e some, se necessário, 0.01 ao resultado

2. Faça um programa capaz de ler 2 números inteiros e calcular seu MDC (Máximo Divisor Comum) e MMC (Mínimo Divisor Comum).

3. Elabore um programa capaz de "ler" duas matrizes e, se possível, dependendo da compatibilidade de dimensões, calcule o produto das mesmas (reveja capítulo 7).

4. Faça um programa que gere na tela a serie de Fibonacci:
0;1;1;2;3;5;8;13;21...

5. Utilizando o programa de animação universal, tente simular a rotação de um objeto na tela:



11 CAPÍTULO

As Interfaces do BASIC com a Linguagem de Máquina

As Interfaces do BASIC com a linguagem de máquina.

Neste ponto do livro, você pode olhar para o teclado do TK e verificar que tudo já foi explicado, com exceção de uma palavra chave e duas funções: POKE (tecla 0), PEEK (tecla 0) e USR (tecla L). Para poder dar uma *noção* do que elas significam, é necessário primeiro explicar sucintamente o que é linguagem de máquina e introduzir o conceito de microprocessador e bytes de memória.

Iremos, inicialmente, introduzir o conceito de *microprocessador*. O computador, como nós o conhecemos, consiste nos seguintes elementos principais:

- microprocessador (modelo Z80, no caso do TK)
- memória
- dispositivos de entrada (teclado, joystick)
- dispositivos de saída (vídeo, impressora)
- memória auxiliar (gravador K7)

O TK pode ser comparado, por analogia, a um ser humano, só que ele "fala" BASIC. Assim, esse "ser" tem um cérebro (que é o microprocessador) e uma memória. Você conversa com ele através do teclado (para enviar informações) e da tela (para receber informações), usando o BASIC. Acontece que o microprocessador *não entende* BASIC... Ele entende uma língua muito mais simples e limitada, à qual chamamos de *linguagem de máquina.*

A linguagem de máquina (também chamada de ASSEMBLY) consiste,assim como o BASIC, em uma série de instruções, sendo que cada uma comanda o microprocessador para que ela faça uma tarefa específica. Entretanto, ela é uma linguagem de baixo nível, o que significa que instruções complicadas como *loops* de FOR/NEXT ou cálculos de funções matemáticas (SIN, COS, EXP, RND, etc.) não são disponíveis, ou seja, pode-se usar apenas instruções que

comandam diretamente o microprocessador.

Mas, o que significa exatamente linguagem de baixo nivel com instruções que comandam diretamente o microprocessador? O problema é que o microprocessador, por ser um circuito eletrônico, só é capaz de entender dois níveis de tensão, chamados, simbolicamente, 0 (zero) e 1 (um). Portanto, somos obrigados a conversar com ele usando códigos formados por cadeias (sequências) de 0s e 1s, onde cada 0 ou 1 é chamado *bit*. Além disso, o microprocessador só é capaz de entender 8 bits (1 byte) por vez.

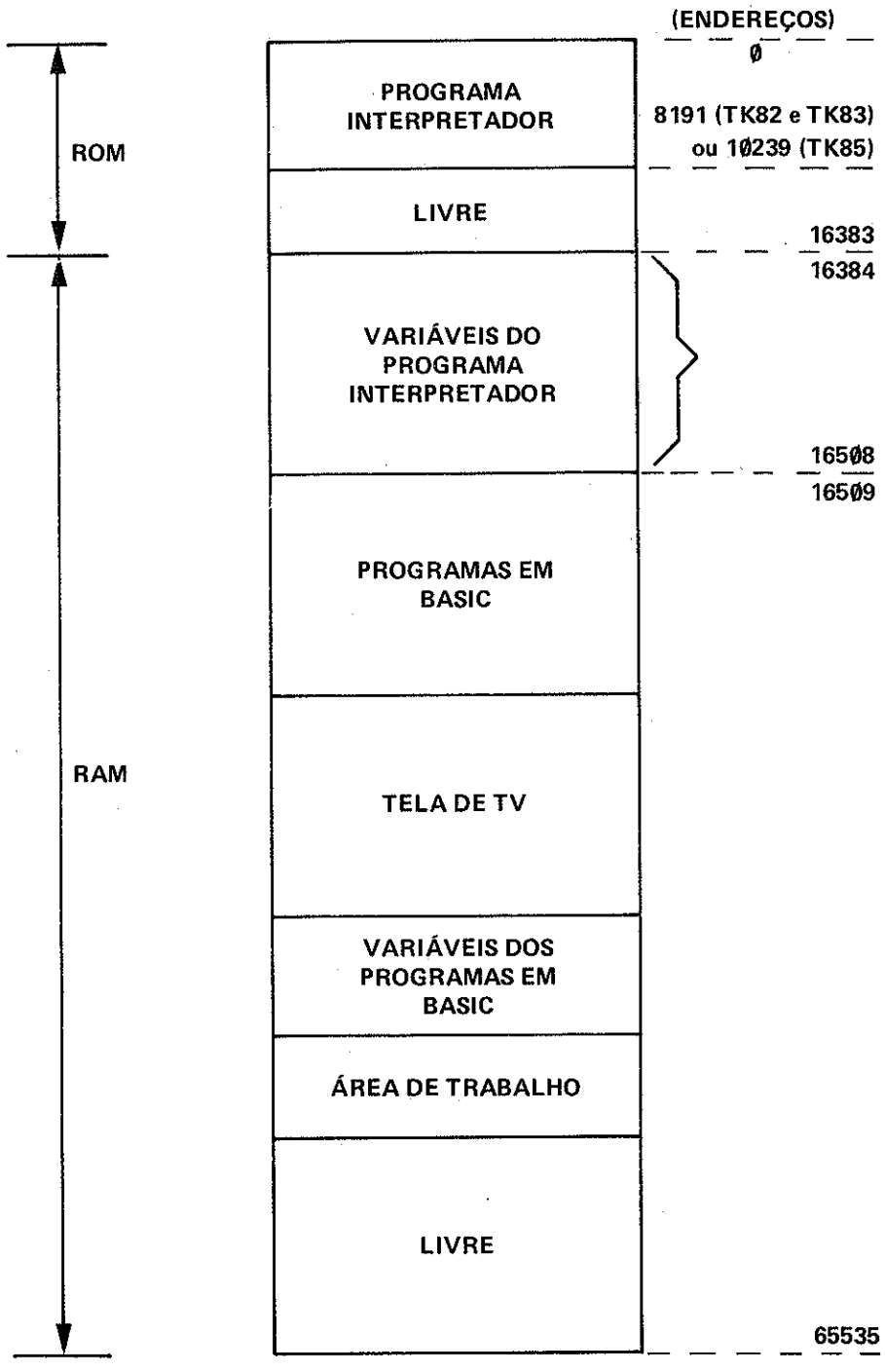
Dessa forma, cada instrução e cada dado deve, obrigatoriamente, consistir em um ou mais grupos de 8 bits. No caso de números, o número máximo que podemos representar em 1 byte é 255. Se o número for maior que 255, ele deverá ser "quebrado" em 2 ou mais bytes para que o microprocessador possa entendê-lo.

Assim como as instruçoes são sempre divididas em grupos de 8 bits, a memória do computador também é didívida em regiões capazes de armazenar apenas 8 bits cada uma.

A memória pode ser imaginada como uma pilha de registradores capazes de armazenar apenas 1 byte cada, sendo que cada registrador tem um endereço que, no caso do TK, pode ir de 0 até 65535. A memória é basicamente dividida em duas partes: a primeira, chamada ROM (Read Only Memory), foi previamente gravada por processos especiais e contém um programa que *interpreta* linguagem BASIC, ou seja, *traduz* as instruções do BASIC para a linguagem de máquina. Esta região da memória é inalterável, não podendo, portanto, ser escrita (apenas lida), e não perde nunca seu conteúdo, mesmo desligando-se o computador. A segunda, chamada RAM (Randon Access Memory), é alterável (pode ser lida ou escrita), e é onde são armazenados o conteúdo da tela de TV, os programas em BASIC, devidamente traduzidos pelo interpretador, (ROM) e as variáveis dos programas.

Ela perde seu conteúdo quando o computador é desligado. Note que o programa interpretador (ou tradutor) que está na memória ROM é escrito em linguagem de máquina, pois o microprocessador deve ser capaz de entendê-lo.

Entretanto,como todo programa, possui variáveis, que devem ser, obrigatoriamente, colocadas na RAM (caso contrário não seriam variáveis); assim, ele reserva para si o início da memória RAM, mais precisamente os endereços 16384 e 16508, para colocar suas próprias variáveis vamos apresentar a seguir um esquema ou "mapa" simplificado da memória do TK (um esquema mais completo pode ser encontrado no próprio manual do TK):



Naturalmente a memória vai até 65535 apenas se tivermos uma expansão de 64k; com 2k de memória, o TK tem memória até o endereço 18431 e com 16k de memória até o endereço 32767.

Poderia surgir agora a seguinte pergunta: existe alguma vantagem em se programar em linguagem de máquina? A resposta é afirmativa: com ela os programas são muito mais rápidos e podemos ter uma visualização e controle maior do sistema; em contrapartida, suas instruções são mais simples e limitadas e exigem um cuidado maior na hora da programação.

Não nos aprofundamos mais neste livro, na linguagem de máquina. Estas idéias foram aqui apresentadas apenas para tentar dar uma noção do que é essa linguagem com o intuito de tentar esclarecer os conceitos que apresentaremos a seguir. Um estudo detalhado da linguagem de máquina é relativamente complexo e o pequeno resumo que apresentamos aqui está longe de mostrar toda a sua potencialidade. Vamos então ao POKE, PEEK e USR.

A palavra-chave POKE permite colocar um número diretamente numa posição qualquer de memória. Naturalmente, este número deve estar compreendido entre 0 e 255 e a posição de memória entre 0 e 65535. Entretanto, as posições de 0 a 16583 são ocupadas por memória ROM que é inalterável ou por espaços vazios; assim, um POKE nessas memórias não produz efeito nenhum. A título de exemplos, faremos dois pequenos truques usando o POKE; pelo "mapa de memória" vimos que os programas em BASIC começam sempre na posição de memória 16509; sabe-se que, normalmente, não é possível em BASIC ter a linha 0. Mas com POKE podemos consegui-la. De fato, experimente digitar:

```
 1 REM PIAZZI-ROSSINI
 2 PRINT " ALL YOU NEED IS LOV
E ".
 3 RUN
```

A seguir, faça o comando direto:

```
POKE 16510,0
```

Agora, faça um LIST e veja o que ocorreu com a linha 1. Tente retirá-la do programa e você verá que não será possível pois o BASIC não está "acostumado" a usar linha 0... Qual a vantagem disto? Ora, você pode fazer um programa, colocar a linha 0 com seu nome e ninguém poderá retirá-lo.

Pode surgir agora a seguinte dúvida: se os programas começam na memória 16509, por que o POKE é feito na memória 16510? A explicação é simples: em BASIC, podemos ter linhas com número maior do que 255; sendo assim, dois bytes são reservados para armazenar o número da linha (no caso

teremos memória 16509 e 16510 ambas com número zero; experimente modificar a memória 16509 usando POKE...). No manual do TK, você poderá encontrar maiores detalhes sobre como cada linha é armazenada na memória, além de explicações de como cada variável é armazenada na memória.

O segundo truque que iremos apresentar possibilita a utilização das linhas 22 e 23 da tela para efeitos da PRINT ou PRINT AT. Na memória 16418 está armazenado um número (inicializado pelo programa interpretador) que diz quantas linhas são reservadas para a região de edição; normalmente, em memória contém o número 2. Basta então alterá-lo para 0; repare:

```
1 POKE 16418,0
2 PRINT AT 22,5;"LOVE IS "
3 PRINT AT 23,5;"THE ANSWER"
```

No entanto, este truque tem uma pequena limitação: não possibilita a utilização da palavra-chave INPUT. Assim, se por acaso você for usá-la, "corrija" antes a memória 16418 com a instrução:

```
POKE 16418,2
```

O que acontece se colocarmos nesta memória um número maior do que 2?

Vamos agora à função PEEK; ela fornece como saída o conteúdo de uma determinada memória (naturalmente entre 0 e 65535). Assim, se ao ligar o computador você fizer:

```
PRINT PEEK 16418
```

Você deverá obter o número 2 na tela pois inicialmente duas linhas são reservadas para a edição. Se o seu programa tiver uma linha 1, o que você espera obter com um PEEK 16510? E com PEEK 16509?

Note que você pode inclusive fazer PEEK de memórias da ROM. De fato para obter o conteúdo da ROM na tela basta fazer:

```
10 SLOW
20 FOR I=0 TO 9000
30 SCROLL
40 PRINT PEEK I
50 NEXT I
```


Como exemplo da função PEEK vamos apresentar um pequeno programa que renumera as linhas de um dado programa em BASIC, sem alterar entretanto os GOTOs ou GOSUBs (isto também seria possível mas o programa é bem mais complexo); a explicação do funcionamento desse programa é razoavelmente complicada e foge dos objetivos desse livro. O programa é aqui apresentado apenas para dar idéia do "potencial" da função PEEK:

```
9000 STOP
9005 REM RENUMBER
9010 FAST
9015 PRINT "NUMERO DA LINHA INIC
IAL "
9020 INPUT NIR
9025 PRINT NIR
9030 PRINT "NOVA LINHA INICIAL "
9035 INPUT NLI
9040 PRINT NLI
9045 PRINT "INTERVALO ENTRE LINH
AS "
9050 INPUT DTL
9055 PRINT DTL
9060 CLS
9065 LET IM=16508
9066 LET NX=PEEK (IM+1)*256+PEEK
 (IM+2)
9067 IF NX<NIR THEN GOTO 9090
9070 IF NX=9000 THEN GOTO 9110
9075 POKE (IM+1),INT (NLI/256)
9080 POKE (IM+2),NLI-256*INT (NL
I/256)
9085 LET NLI=NLI+DTL
9090 LET IM=IM+5
9095 IF PEEK IM=118 THEN GOTO 90
66
9100 LET IM=IM+1
9105 GOTO 9095
9110 LIST
```

Coloque seu programa antes da linha 9000 e forneça ao RENUMBER a linha inicial de seu programa,a nova linha inicial e a distância que você deseja entre as linhas. Este programa é muito útil quando não há mais "espaço" entre as linhas para colocar novas instruções, ou para tornar as listagens mais "elegantes"

Para finalizar o livro, vamos dar uma idéia sobre a função USR; ela seria como um GOSUB só que em vez de "ir" para uma linha de programa ela vai direto para uma posição de memória executar uma subrotina escrita em linguagem de máquina. Apenas para dar uma demonstração da potencialidade da linguagem de máquina, experimente o seguinte programa que "enche" a tela um determinado caractere:

```
1 REM YWX/X
2 POKE 16516,215
3 POKE 16516,253
4 LET T=USR 16514
```

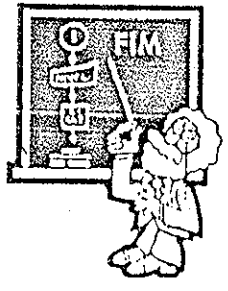
Digite RUN e NEW LINE... A seguir, LIST e NEW LINE; veja o que aconteceu com os caracteres na linha 1...

Os caracteres na linha REM significam um pequeno programa em linguagem de máquina; o caractere que é colocado na tela é o segundo caractere da linha REM.

Experimente modificá-lo... Compare a velocidade deste programa (mesmo em SLOW) com o seguinte programa em BASIC:

```
3 PRINT "W"
4 GOTO 3
```

Isto é tudo, por enquanto. No segundo volume desta série, nos aprofundaremos no BASIC dando ênfase à estrutura de programação.



Querendo adquirir um destes livros escreva para.

ALEPH
PUBLICAÇÕES E ASSESSORIA PEDAGÓGICA LTDA
Av. Brig. Faria Lima, 1451 - conj. 31
01451 - São Paulo - SP - Tel.: (011) 813-4555

BASIC TK - Volume II

Pierluigi Piazzi
Flávio Rossini

- A continuação de um dos maiores sucessos ed
  toriais em Informática.
- Aprofundamento do BASIC com estrutura de pr
  gramação.
- Truques utilizando o sistema operacional dos m
  cros da linha SINCLAIR (ZX-81, TK 82/83/85, CI
  200, RINGO).

Uma obra elaborada com os mesmos cuidados d
didáticos do VOLUME I.

DISSECANDO JOGOS (em Basic TK - comentado linha por linha)

Carlos
Eduardo Rocha Salvato

Os sete jogos contidos neste volume têm uma ca
racterística única: o leitor participa do processo de
criação acompanhando a montagem linha por linha.

Modificações e aperfeiçoamentos são propostos e
resolvidos com detalhes. Ao terminar este livro o leitor
além de ter sete jogos geniais, terá adquirido a habili-
dade de programar com criatividade.

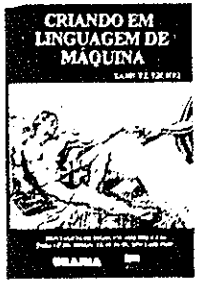
Uma obra didática indispensável ao usuário de um
micro da linha SINCLAIR: Ringo R-470, CP-200, TK-
82/83/85, AS-1000, SINCLAIR ZX-81 ou TS-1000.

JOGOS EM LINGUAGEM DE MÁQUINA - Volume III

Selecionados por
Pierluigi Piazzi

Continuando a série, mais jogos inéditos
INVASÃO
IRA
VERMES DE AREIA (Duna)
BASQUETE
FROGGIE e muitos outros

- Desenvolva o lado direito de seu cérebro, (visão
  espacial), jogando o fascinante DEDALO
- A digitação não exige conhecimentos de Lingua
  gem de Máquina.
- Jogos em ASSEMBLY Z 80 para computadore
  compatíveis com SINCLAIR ZX-81 (TK 82/83/85
  CP 200, TS 1000 etc.).



CRIANDO EM LINGUAGEM DE MÁQUINA

Samuel Ejchel

Não basta se conhecer as instruções do AS-
SEMBLY para se criar um programa interessante, exis-
tem truques e "dicas" indispensáveis para o bom pro
gramador:

Gerenciando a tela
Utilizando o teclado
Usando o "Joystick"
Gerando movimentos multiplos
Criando opções
Implementando requintes

Em cada capítulo um jogo original, totalmente ex
plicado, ilustrando os truques de programação utiliza
dos — Uma obra indispensável para quem programa
em micros da linha SINCLAIR: CP 200 - RINGO - TK
82/83/85 - AS 1000.

Este livro foi impresso na
Gráfica Palas Athena
Associação "Palas Athena" do Brasil
Rua Dona Ana Nery, 846
Fone: 279-6288 — CEP 01522
Cambuci — São Paulo